RAIRO-Theor. Inf. Appl. 55 (2021) 9
https://doi.org/10.1051/ita/2021003

RAIRO - Theoretical Informatics and Applications
www.rairo-ita.org

ON RESTARTING AUTOMATA WITH AUXILIARY SYMBOLS AND SMALL WINDOW SIZE*

František Mráz[1] and Friedrich Otto[2,**]

Abstract. Here we show that for monotone RWW- (and RRWW-) automata, window size two is sufficient, both in the nondeterministic as well as in the deterministic case. For the former case, this is done by proving that each context-free language is already accepted by a monotone RWW-automaton of window size two. In the deterministic case, we first prove that each deterministic pushdown automaton can be simulated by a deterministic monotone RWW-automaton of window size three, and then we present a construction that transforms a deterministic monotone RWW-automaton of window size three into an equivalent automaton of the same type that has window size two. Furthermore, we study the expressive power of shrinking RWW- and RRWW-automata the window size of which is just one or two. We show that for shrinking RRWW-automata that are nondeterministic, window size one suffices, while for nondeterministic shrinking RWW-automata, we already need window size two to accept all growing context-sensitive languages. In the deterministic case, shrinking RWW- and RRWW-automata of window size one accept only regular languages, while those of window size two characterize the Church-Rosser languages.

Mathematics Subject Classification. 68Q45, 68Q42.

Received December 6, 2019. Accepted April 29, 2021.

1. Introduction

The restarting automaton was introduced in [6] as a formal model for the linguistic technique of 'analysis by reduction'. A restarting automaton, RRWW-automaton for short, is a device M that consists of a finite-state control, a flexible tape containing a word delimited by sentinels, and a read/write window of fixed finite size. This window is moved along the tape by move-right steps until the control decides (nondeterministically) that the contents of the window should be rewritten by some *shorter* word. In fact, the new word may contain auxiliary symbols that do not belong to the input alphabet. After a rewrite, M can continue to move its window until it either halts and accepts, or halts and rejects, or restarts, which means that it places its window over the left end of the tape and reenters its initial state. It follows that each computation of M can be described through a sequence of cycles and a tail computation. Here a cycle is a part of a computation that begins after a restart step (or by the first step from an initial configuration) and that ends with the next restart step, and

*Some of the results of this paper have been announced at NCMA 2019 in Valencia, Spain, July 2019, and at DLT 2019 in Warsaw, Poland, August 2019. Extended abstracts can be found in the proceedings of these conferences [20, 21].

Keywords and phrases: Restarting automaton, window size, weight function, language class.

[1] Charles University, Faculty of Mathematics and Physics, Malostranské nám. 25, 118 25 Prague 1, Czech Republic.
[2] Fachbereich Elektrotechnik/Informatik, Universität Kassel, 34109 Kassel, Germany.

** Corresponding author: f.otto@uni-kassel.de

the tail is the part of a computation that begins after the last restart step, that is, it ends with either an accept step or with M getting stuck in a configuration to which no transition applies.

By requiring that an RRWW-automaton always performs a restart step immediately after executing a rewrite operation, we obtain the so-called RWW-automaton. Within any cycle, such an automaton cannot scan the suffix of the tape contents that is to the right of the position at which the rewrite operation is performed. Although the definition of the RWW-automaton is clearly much more restricted than that of the RRWW-automaton, it is a long-standing open problem whether the class of languages $\mathcal{L}(\mathsf{RWW})$ accepted by RWW-automata coincides with the class of languages $\mathcal{L}(\mathsf{RRWW})$ accepted by RRWW-automata.

In [7], it was shown that the monotone variants of the nondeterministic RWW- and RRWW-automaton (see Sect. 2 for the definitions) accept exactly the class CFL of context-free languages, while the corresponding deterministic automata characterize the class DCFL of deterministic context-free languages. As a restarting automaton has a read/write window of a fixed finite, but otherwise arbitrary, size $k \geq 1$, it is only natural to ask for the influence that the window size has on the expressive power of the various types of restarting automata. In [18], it is shown that for those types of restarting automata that do not admit auxiliary symbols, the window size yields infinite ascending hierarchies of language classes. In fact, the increase in the descriptional complexity when turning a restarting automaton without auxiliary symbols that has window size $k+1$ into an equivalent automaton of window size k can in general not even be bounded from above by any recursive function [12].

On the other hand, in [29], Natalie Schluter presents a construction that allows to simulate a monotone RRWW-automaton of window size $k \geq 3$ by an automaton of the same type that has only window size $k-1$. In addition, she presents a construction that shows that each monotone RRWW-automaton of window size $k \geq 2$ can be simulated by a monotone RWW-automaton of window size k. This implies that the context-free languages are already accepted by monotone RWW-automata of window size two.

By combining these constructions, one can transform a given monotone RWW-automaton M_k of window size $k > 2$ into an equivalent monotone RWW-automaton M_2 of window size two. Indeed, one can easily interpret M_k as a monotone RRWW-automaton of window size k. Then one can iteratively apply the former of Natalie Schluter's constructions $k-2$ times to obtain a monotone RRWW-automaton M_2' of window size two, and finally, one can apply the latter of her constructions to get the monotone RWW-automaton M_2. However, both of Natalie Schluter's constructions are technically quite involved and therefore not easy to follow. Here we describe a direct way of accepting any context-free language by a monotone RWW-automaton of window size two by presenting a simulation of the derivations of a context-free grammar in Chomsky normal form by a monotone RWW-automaton of window size two.

Next we turn to the deterministic case. Here we first present a construction that turns a deterministic pushdown automaton (PDA) into a deterministic monotone RWW-automaton of window size three that accepts the same language. Then we describe a construction that shows how to simulate a deterministic monotone RWW-automaton of window size three by a deterministic monotone RWW-automaton of window size two. Here the crucial point consists in observing that each deterministic RWW-automaton of window size two is already monotone. Together our results on the deterministic case show that the class DCFL is actually already characterized by deterministic monotone RWW-automata of window size two. It is well known that RWW-automata of window size one only accept regular languages [18]. Thus, based on the window size we have just two classes of languages that are accepted by monotone RWW-automata, both in the nondeterministic as well as in the deterministic case.

In order to investigate the relationship between RWW- and RRWW-automata, a generalization of the restarting automaton, called *shrinking restarting automaton*, was introduced in [10]. A shrinking restarting automaton M is defined just like an RRWW-automaton with the one exception that it is no longer required that each rewrite step $u \to v$ of M must be length-reducing. Instead, there must exist a weight function ω that assigns a positive integer $\omega(a)$ to each letter a of M's tape alphabet Γ such that, for each rewrite step $u \to v$ of M, $\omega(u) > \omega(v)$ holds, where the function ω is extended to a morphism $\omega : \Gamma^* \to \mathbb{N}$ as usual.

Here we also study the expressive power of shrinking RWW- and RRWW-automata of window size one and two. We shall see that nondeterministic shrinking RRWW-automata of window size one are already as

expressive as nondeterministic shrinking RRWW-automata in general, which are known to be equivalent to the finite-change automata of [30]. On the other hand, for nondeterministic shrinking RWW-automata, window size one is not enough, as we already need window size two to accept all growing context-sensitive languages. However, window size nine suffices to again obtain all language accepted by finite-change automata. In fact, it remains open whether window size nine is the smallest possible, that is, whether nondeterministic shrinking RWW-automata of window size eight are really less expressive than those of window size nine.

Finally, we consider shrinking RWW-automata and monotone shrinking RRWW-automata of window size one and their deterministic variants. It turns out that deterministic shrinking RWW- and RRWW-automata of window size one just accept the regular languages, while with window size two, these automata characterize the Church-Rosser languages. Furthermore, also the monotone shrinking RWW-automata of window size one just accept the regular languages, while shrinking RWW-automata and monotone shrinking RRWW-automata of window size one are strictly more expressive.

This paper is structured as follows. After presenting the necessary definitions and notation in Section 2, we restate in short the known results on the influence of the window size on the expressive power of restarting automata. In Section 3, we then concentrate on nondeterministic monotone RWW- and RRWW-automata, in Section 4, we study deterministic monotone RWW- and RRWW-automata, and we discuss the influence of the window size on deterministic RWW- and RRWW-automata that are non-monotone. Then we turn to shrinking restarting automata in Section 5, in which we investigate the expressive power of nondeterministic shrinking RWW- and RRWW-automata of window size one and two. Finally, in Section 6, we consider the deterministic case. The paper closes with Section 7 in which we summarize our results using a diagram that depicts the relationships between the various language classes considered and state a number of open problems.

2. Definitions and notation

Throughout the paper, λ will denote the empty word, and $\mathbb{N}_+$ will denote the set of all positive integers, while $\mathbb{N}$ is used to denote the set of all non-negative integers. Furthermore, for any type of automaton X, we use the notation $\mathcal{L}(\mathsf{X})$ to denote the class of languages that are accepted by automata of type X.

Definition 2.1. A (one-way) *restarting automaton*, RRWW-automaton for short, is a one-tape machine that is described by an 8-tuple $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite tape alphabet containing Σ, where the symbols in $\Gamma \setminus \Sigma$ are called *auxiliary symbols*, the symbols $¢, \$ \notin \Gamma$, called *sentinels*, serve as markers for the left and right border of the workspace, respectively, $q_0 \in Q$ is the initial state, $k \geq 1$ is the size of the *read/write window*, and

$$\delta : Q \times \mathcal{PC}^{(k)} \to \mathcal{P}((Q \times (\{\mathsf{MVR}\} \cup \mathcal{PC}^{\leq (k-1)})) \cup \{\mathsf{Restart}, \mathsf{Accept}\})$$

is the *transition relation*. Here $\mathcal{P}(S)$ denotes the powerset of the set S, $\mathcal{PC}^{(k)}$ is the set of *possible contents* of the read/write window of M, where

$$\mathcal{PC}^{(i)} := (\{¢\} \cdot \Gamma^{i-1}) \cup \Gamma^i \cup (\Gamma^{\leq i-1} \cdot \{\$\}) \cup (\{¢\} \cdot \Gamma^{\leq i-2} \cdot \{\$\}) \quad (i \geq 0),$$

and

$$\Gamma^{\leq n} := \bigcup_{i=0}^{n} \Gamma^i \quad \text{and} \quad \mathcal{PC}^{\leq (k-1)} := \bigcup_{i=0}^{k-1} \mathcal{PC}^{(i)}.$$

The transition relation contains four different types of transition steps:

1. A *move-right step* is of the form $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u \in \mathcal{PC}^{(k)}$, $u \neq \$$. If M is in state q and sees the word u in its read/write window, then this move-right step causes M to shift the read/write

window one position to the right and to enter state q'. However, if the contents u of the read/write window is only the symbol \$, then no shift to the right is possible.

2. A *rewrite step* is of the form $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u \in \mathcal{PC}^{(k)}$, $u \neq \$$, and $v \in \mathcal{PC}^{\leq(k-1)}$ such that $|v| < |u|$. It causes M to replace the contents u of the read/write window by the word v, thereby shortening the tape, and to enter state q'. Furthermore, the read/write window is placed immediately to the right of the word v. However, some additional restrictions apply in that the sentinels ¢ and \$ must not disappear from the tape nor that new occurrences of these symbols are created. In addition, the read/write window must not move across the right sentinel \$, that is, if the word u ends in \$, then so does the word v, and after performing the rewrite operation, the read/write window is placed on the \$-symbol.
3. A *restart step* is of the form $\mathsf{Restart} \in \delta(q, u)$, where $q \in Q$ and $u \in \mathcal{PC}^{(k)}$. It causes M to place the read/write window over the left end of the tape, so that the first symbol it contains is the left sentinel ¢, and to reenter the initial state q_0.
4. An *accept step* is of the form $\mathsf{Accept} \in \delta(q, u)$, where $q \in Q$ and $u \in \mathcal{PC}^{(k)}$. It causes M to halt and accept.

If $\delta(q, u) = \emptyset$ for some $q \in Q$ and $u \in \mathcal{PC}^{(k)}$, then M necessarily halts when it is in state q with the word u in its window, and we say that M *rejects* in this situation. There is one additional restriction that the transition relation must satisfy. This restriction says that, when ignoring move operations, *rewrite steps and restart steps alternate* within any computation of M, with a rewrite step coming first.

Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an RRWW-automaton. A *configuration* of M is described by a word of the form $\alpha q \beta$, where $q \in Q$, and either $\alpha = \lambda$ and $\beta \in \{¢\} \cdot \Gamma^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot \Gamma^*$ and $\beta \in \Gamma^* \cdot \{\$\}$; here q represents the current state, $\alpha\beta$ is the current contents of the tape, and it is understood that the window contains the first k symbols of β or all of β when $|\beta| \leq k$. By $\vdash_M$ we denote the *single-step computation relation* that M induces on its set of configurations, and $\vdash_M^*$ is the *computation relation* of M, which is the reflexive and transitive closure of $\vdash_M$. A *restarting configuration* is of the form $q_0 ¢ w \$$, where $w \in \Gamma^*$; if $w \in \Sigma^*$, then $q_0 ¢ w \$$ is an *initial configuration*. Thus, initial configurations are a particular type of restarting configurations.

A phase of a computation of M, called a *cycle*, begins with a restarting configuration. The window is moved along the tape by MVR operations and a single rewrite operation until a restart operation is performed and, thus, a new restarting configuration is reached. Accordingly, a computation consists of a sequence of cycles that is followed by a *tail*, which is the part of a computation that comes after the last restart operation. By $\vdash_M^c$ we denote the relation on restarting configurations that is induced through the execution of a cycle, and we use $\vdash_M^{c^*}$ to denote its reflexive and transitive closure. The above restriction on the transition relation implies that M performs *exactly one* rewrite operation during each cycle – thus each new phase starts on a shorter word than the previous one – and that it executes at most one rewrite operation during a tail computation.

An input word $w \in \Sigma^*$ is *accepted* by M, if there is a computation which, starting with the initial configuration $q_0 ¢ w \$$, finishes by executing an accept step. By $L(M)$ we denote the language consisting of all (input) words accepted by M; we say that M *accepts the language* $L(M)$.

In general, an RRWW-automaton is nondeterministic, that is, for some pairs (q, u), there may be more than one applicable transition step. If this is not the case, then the automaton is deterministic. We use the prefix det- to denote classes of deterministic restarting automata.

Many restricted classes of restarting automata have been introduced and studied. An *RWW-automaton* is an RRWW-automaton that is required to execute a restart step immediately after performing a rewrite step. Accordingly, for RWW-automata we combine a rewrite step with the subsequent restart step into a single *combined rewrite/restart step* to simplify the notation.

- A *combined rewrite/restart step* is of the form $v \in \delta(q, u)$, where $q \in Q$, $u \in \mathcal{PC}^{(k)}$, $u \neq \$$, and $v \in \mathcal{PC}^{\leq(k-1)}$ such that $|v| < |u|$. It causes M to replace the contents u of the read/write window by the word v, thereby shortening the tape, to place the read/write window over the left end of the tape, so that the first symbol it sees is the left sentinel ¢, and to reenter the initial state q_0.

An *RRW-automaton* is an RRWW-automaton which does not use any auxiliary symbols, that is, its tape alphabet coincides with its input alphabet. An *RR-automaton* is an RRW-automaton the rewrite steps of which can be viewed as deletions, that is, if $(q', v) \in \delta(q, u)$, then v is a scattered subword of u. Obviously, the restrictions on the rewrite operations can be combined with the restriction on the restart operations (requiring to restart immediately after each rewrite step), which leads to the RW- and the R-automaton. The size of the read/write window is an essential parameter of a restarting automaton. For each $k \geq 1$, we use the notation $\mathsf{X}(k)$ to denote those restarting automata of type X that have a read/write window of size k.

Recall that the computation of a restarting automaton proceeds in cycles, where each cycle contains exactly one rewrite step. Thus, each cycle C contains a unique configuration of the form $\alpha q \beta$ in which a rewrite step is applied. Now $|\beta|$ is called the *right distance* of C, which is denoted by $D_r(C)$.

A sequence of cycles $S = (C_1, C_2, \ldots, C_n)$ is called *monotone* if $D_r(C_1) \geq D_r(C_2) \geq \cdots \geq D_r(C_n)$. A computation is *monotone* if the corresponding sequence of cycles is monotone. Observe that here the tail of the computation is not taken into account. Finally, a restarting automaton is called *monotone* if all its computations that start with an initial configuration are monotone. The prefix mon- is used to denote the various classes of monotone restarting automata.

The following result extends the characterization of the context-free languages by monotone restarting automata that is presented in [7] to window size three.

Theorem 2.2. [23] $\mathsf{CFL} = \mathcal{L}(\mathsf{mon\text{-}RRWW}(3)) = \mathcal{L}(\mathsf{mon\text{-}RWW}(3))$.

Concerning the monotone deterministic restarting automata, the following results have been derived in [7].

Theorem 2.3.
$$\begin{aligned} \mathsf{DCFL} &= \mathcal{L}(\mathsf{det\text{-}mon\text{-}RRWW}) = \mathcal{L}(\mathsf{det\text{-}mon\text{-}RWW}) \\ &= \mathcal{L}(\mathsf{det\text{-}mon\text{-}RRW}) = \mathcal{L}(\mathsf{det\text{-}mon\text{-}RW}) \\ &= \mathcal{L}(\mathsf{det\text{-}mon\text{-}RR}) = \mathcal{L}(\mathsf{det\text{-}mon\text{-}R}). \end{aligned}$$

On the other hand, it is easily seen that each RRWW-automaton (RWW-automaton) with a read/write window of size one is just an RR-automaton (R-automaton), as its rewrite operations must be length-reducing. Concerning window size one, the following results are known. Here REG denotes the class of regular languages.

Theorem 2.4. [13, 18]
(a) $\mathsf{REG} = \mathcal{L}((\mathsf{mon\text{-}})\mathsf{RWW}(1)) = \mathcal{L}(\mathsf{det\text{-}(mon\text{-})RRWW}(1))$.
(b) $\mathsf{REG} \subsetneq \mathcal{L}(\mathsf{mon\text{-}RRWW}(1)) \subsetneq \mathsf{CFL}$.

Thus, window size one restricts the expressive power of monotone deterministic and nondeterministic RWW- and RRWW-automata considerably.

In [29], Natalie Schluter established the following results concerning the window size of monotone restarting automata. These results were already announced in [27, 28].

Theorem 2.5. *For all $k \geq 2$,*
(a) $\mathcal{L}(\mathsf{mon\text{-}RWW}(k)) = \mathcal{L}(\mathsf{mon\text{-}RRWW}(k))$.
(b) $\mathcal{L}(\mathsf{det\text{-}mon\text{-}RWW}(k)) = \mathcal{L}(\mathsf{det\text{-}mon\text{-}RRWW}(k))$.

Theorem 2.6. *For all $k \geq 2$, $\mathcal{L}(\mathsf{mon\text{-}RRWW}(k+1)) = \mathcal{L}(\mathsf{mon\text{-}RRWW}(k))$.*

As a consequence, we obtain that $\mathsf{CFL} = \mathcal{L}(\mathsf{mon\text{-}RRWW}(2)) = \mathcal{L}(\mathsf{mon\text{-}RWW}(2))$. Unfortunately, the constructions used by Natalie Schluter to prove her results are technically very involved and, accordingly, very hard to follow. In the next section, we provide a direct and much simpler proof for the characterization of the class CFL by monotone restarting automata of window size two.

3. On nondeterministic monotone RWW-automata

Our first main result states that each context-free language is accepted by an RWW-automaton of window size two.

Theorem 3.1. $\mathsf{CFL} = \mathcal{L}(\mathsf{mon\text{-}RWW}(2))$.

Proof. Obviously, $\mathcal{L}(\mathsf{mon\text{-}RWW}(2)) \subseteq \mathcal{L}(\mathsf{mon\text{-}RWW}(3))$. Hence, because of Theorem 2.2, it remains to show that each context-free language is accepted by a monotone RWW-automaton of window size two.

Let $L \subseteq \Sigma^*$ be a context-free language, and let $G = (N, \Sigma, S, P)$ be a context-free grammar for L. Without loss of generality, we may assume that the production rules in P have the following form, which is easily obtained from the Chomsky normal form:

$$\begin{array}{lll} (1) & S \to \lambda, & \text{if } \lambda \in L, \\ (2) & S \to a, & \text{if } a \in \Sigma \cap L, \\ (3) & A \to BC, & \text{where } A \in N \text{ and } B, C \in (N \setminus \{S\}) \cup \Sigma. \end{array}$$

Below we use the notation $\Rightarrow_G^*$ to denote the derivation relation that is induced by the productions of G on the sentential forms $(N \cup \Sigma)^*$.

Now we define an RWW-automaton $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, 2, \delta)$, where $Q = \{q_0, q, q_+\}$ and

$$\Gamma = N \cup \Sigma \cup \{\, (A, B) \mid A, B \in N \cup \Sigma \,\},$$

and we define a morphism $\pi\colon \Gamma^* \to (N \cup \Sigma)^*$ by taking

$$\pi(A) = A \text{ and } \pi((A, B)) = AB \text{ for all } A, B \in N \cup \Sigma.$$

Thus, M uses the nonterminals A of G and the pairs of the form (A, B) with $A, B \in N \cup \Sigma$ as auxiliary symbols that represent the words A and AB, respectively. The transition relation δ of M is defined as follows, where $A, B, C \in N \cup \Sigma$ and $X \in N$:

$$\begin{array}{lll} (1) & \delta(q_0, ¢\$) = \{\mathsf{Accept}\} & \text{if } (S \to \lambda) \in P, \\ (2) & \delta(q_0, ¢\gamma) \ni (q_+, \mathsf{MVR}) & \text{for } \gamma \in \Gamma \text{ such that } S \Rightarrow_G^* \pi(\gamma), \\ (3) & \delta(q_+, \gamma\$) = \{\mathsf{Accept}\} & \text{for } \gamma \in \Gamma \text{ such that } S \Rightarrow_G^* \pi(\gamma), \\ (4) & \delta(q_0, ¢\gamma) \ni (q, \mathsf{MVR}) & \text{for all } \gamma \in \Gamma, \\ (5) & \delta(q, AB) \ni (A, B) & \text{for all } A, B \in N \cup \Sigma, \\ (6) & \delta(q, AB) \ni C & \text{if } (C \to AB) \in P, \\ (7) & \delta(q, (A, B)\gamma) \ni (q, \mathsf{MVR}) & \text{for } \gamma \in \Gamma, \\ (8) & \delta(q, (A, B)C) \ni (A, X) & \text{if } (X \to BC) \in P, \\ (9) & \delta(q, (A, B)C) \ni X & \text{if } (X \to AZ), (Z \to BC) \in P \text{ for some } Z \in N. \end{array}$$

Obviously, M is an RWW-automaton of window size two. The encoding of two letters $A, B \in N \cup \Sigma$ as a single letter (A, B) is used to mark the position in the actual sentential form that has to be examined in the next step. Thus, the tape contents always consists of a sequence of pairs followed by a sequence of letters from $N \cup \Sigma$, where any of these two subsequences can be empty. Since each rewrite step is performed at the border between these two subsequences, it is easily seen that M is monotone.

It remains to verify that M accepts the context-free language L. It is easily seen that an accepting computation of M constructs a G-derivation for the given input in reverse order. Hence, we have $L(M) \subseteq L$.

On the other hand, let $w \in L$. If $w = \lambda$, then $w \in L(M)$. So assume that $|w| \geq 1$. Then there exists a rightmost G-derivation

$$S \Rightarrow_G \alpha_1 \Rightarrow_G \alpha_2 \Rightarrow_G \cdots \Rightarrow_G \alpha_n = w.$$

Claim. For all $i = 1, 2, \ldots, n$, $q_0 ¢ \alpha_i \$ \vdash_M^* \mathsf{Accept}$.

Proof. At first, we prove the claim for $n = 1$, then for any $n > 1$. If $n = 1$, then $|w| \leq 2$ and $\alpha_1 = w$. For $|w| = 1$, we have

$$q_0 ¢ \alpha_1 \$ \vdash_M ¢ q_+ \alpha_1 \$ \vdash_M \mathsf{Accept}$$

by rules (2) and (3), and for $|w| = 2$, we have $w = ab$ for some letters $a, b \in \Sigma$ and

$$q_0 ¢ \alpha_1 \$ = q_0 ¢ ab \$ \vdash_M ¢ q a b \$ \vdash_M q_0 ¢ (a, b) \$ \vdash_M ¢ q_+ (a, b) \$ \vdash_M \mathsf{Accept}$$

by rules (4), (5), (2), and (3).

For $n > 1$, we proceed by induction on i. If $i = 1$, then $\alpha_1 = AB$ for some $A, B \in (N - \{S\}) \cup \Sigma$ and $(S \to AB) \in P$, and

$$q_0 ¢ \alpha_1 \$ = q_0 ¢ AB \$ \vdash_M ¢ q A B \$ \vdash_M q_0 ¢ (A, B) \$ \vdash_M ¢ q_+ (A, B) \$ \vdash_M \mathsf{Accept}$$

by rules (4), (5), (2), and (3).

Now assume that $q_0 ¢ \alpha_j \$ \vdash_M^* \mathsf{Accept}$ has already been shown for all $j \leq i$ and some i, $1 \leq i \leq n - 1$. We consider the sentential form α_{i+1}.

If $i = 1$, then either $\alpha_1 = Ab \Rightarrow_G B_1 B_2 b = \alpha_2$ for some $A \in N$, $b \in \Sigma$, and $(A \to B_1 B_2) \in P$, or $\alpha_1 = BA \Rightarrow_G BB_2B_3$ for some $A \in N$, $B \in (N \cup \Sigma)$, and $(A \to B_2 B_3) \in P$. There is no other possibility, as we consider a rightmost derivation. In the former case,

$$q_0 ¢ \alpha_2 \$ = q_0 ¢ B_1 B_2 b \$ \vdash_M ¢ q B_1 B_2 b \$ \vdash_M q_0 ¢ A b \$ = q_0 ¢ \alpha_1 \$$$

by rules (4) and (6), which implies $q_0 ¢ \alpha_2 \$ \vdash_M^* \mathsf{Accept}$ by the induction hypothesis, while in the latter case,

$$\begin{array}{lcllll} q_0 ¢ \alpha_2 \$ & = & q_0 ¢ B B_2 B_3 \$ & \vdash_M & ¢ q B B_2 B_3 \$ & \vdash_M \quad q_0 ¢ (B, B_2) B_3 \$ \\ & \vdash_M & ¢ q (B, B_2) B_3 \$ & \vdash_M & q_0 ¢ (B, A) \$ & \end{array}$$

by rules (4), (5), and (8), which yields $q_0 ¢ \alpha_2 \$ \vdash_M^* \mathsf{Accept}$ by rules (2) and (3).

Finally, assume that $i \geq 2$. Then $\alpha_{i-1} \Rightarrow_G \alpha_i \Rightarrow_G \alpha_{i+1}$ is a part of the above rightmost derivation. The latter step implies that

$$\alpha_i = B_1 B_2 \cdots B_m A a_{m+3} \cdots a_{r-1} a_r$$

for some $B_1, B_2, \ldots, B_m \in N \cup \Sigma$, a nonterminal $A \in N$, terminals $a_{m+3}, \ldots, a_{r-1}, a_r \in \Sigma$, and a production $(A \to B_{m+1} B_{m+2}) \in P$ such that

$$\alpha_{i+1} = B_1 \cdots B_m B_{m+1} B_{m+2} a_{m+3} \cdots a_r.$$

Furthermore, $\alpha_{i-1} = XCY \Rightarrow_G XDEY = \alpha_i$ for a production $(C \to DE) \in P$ and some words $X \in (N \cup \Sigma)^*$ and $Y \in \Sigma^*$. As $\alpha_{i-1} \Rightarrow_G \alpha_i$ is a part of the above rightmost derivation, it follows that either the distinguished occurrence of the nonterminal A in α_i is produced by the above application of the production $(C \to DE)$ implying that $DE = B_m A$ or $DE = A a_{m+3}$, or that $X = B_1 B_2 \cdots B_m A a_{m+3} \cdots a_{m+s-1}$, $DE = a_{m+s} a_{m+s+1}$, and $Y = a_{m+s+2} a_{m+s+3} \cdots a_r$ for some $s \geq 3$.

Using instructions (4), (5), and (7), M can convert the configuration

$$q_0 ¢ \alpha_{i+1} \$ = q_0 ¢ B_1 \cdots B_m B_{m+1} B_{m+2} a_{m+3} \cdots a_r \$$$

into the configuration

$$¢(B_1, B_2)\cdots(B_{m-1}, B_m)qB_{m+1}B_{m+2}a_{m+3}\cdots a_r\$ \tag{3.1}$$

or

$$¢(B_1, B_2)\cdots(B_{m-2}, B_{m-1})q(B_m, B_{m+1})B_{m+2}a_{m+3}\cdots a_r\$ \tag{3.2}$$

depending on the parity of the index m.

In the former case, $B_{m+1}B_{m+2}$ is then rewritten into the nonterminal A by rule (6), and as the derivation step $\alpha_{i-1} \Rightarrow_G \alpha_i$ either produces the displayed nonterminal A or occurs in the suffix following this letter, we can conclude that the computation $q_0¢\alpha_i\$ \vdash_M^* \mathsf{Accept}$ starts with the encoding of the prefix $B_1B_2\cdots B_m$ into the word $(B_1, B_2)\cdots(B_{m-1}, B_m)$. Hence, the configuration obtained in one step from (3.1) appears in an accepting computation that starts from $q_0¢\alpha_i\$$ and $q_0¢\alpha_{i+1}\$ \vdash_M^* \mathsf{Accept}$ follows.

In the latter case, $(B_m, B_{m+1})B_{m+2}$ is either rewritten into (B_m, A) by rule (8) or into C by rule (9), if $DE = B_mA$. Here it follows analogously that the computations $q_0¢\alpha_i\$ \vdash_M^* \mathsf{Accept}$ and $q_0¢\alpha_{i-1}\$ \vdash_M^* \mathsf{Accept}$ start with encoding the prefix $B_1B_2\cdots B_mA$ into the word $(B_1, B_2)\cdots(B_m, A)$ or $B_1B_2\cdots B_{m-1}$ into the word $(B_1, B_2)\cdots(B_{m-2}, B_{m-1})$, respectively. Hence, the configuration obtained in one step from (3.2) appears in an accepting computation starting either from $q_0¢\alpha_i\$$ or from $q_0¢\alpha_{i-1}\$$. Therefore, also in these cases $q_0¢\alpha_{i+1}\$ \vdash_M^* \mathsf{Accept}$ follows. □

For $i = n$, the claim above yields that $q_0¢\alpha_n\$ = q_0¢w\$ \vdash_M^* \mathsf{Accept}$, which means that $w \in L(M)$. Thus, in summary, we have shown that $L(M) = L$. □

As each RWW(2)-automaton can also be interpreted as an RRWW(2)-automaton, the above result yields the following consequence.

Corollary 3.2. $\mathsf{CFL} = \mathcal{L}(\mathsf{mon\text{-}RRWW}(2))$.

From a given monotone RWW-automaton M of window size $k \geq 3$, we can first construct a pushdown automaton P for the language $L(M)$ as shown in the proof of Theorem 2.2 in [7]. From this PDA, we can derive a context-free grammar G in Chomsky normal form for $L(G)$, and then, using the construction above, we obtain a monotone RWW-automaton of window size two for $L(M)$. Furthermore, as for all $k \geq 3$, $\mathcal{L}(\mathsf{mon\text{-}RWW}(2)) \subseteq \mathcal{L}(\mathsf{mon\text{-}RWW}(k))$, we see that, for each $k \geq 3$,

$$\mathcal{L}(\mathsf{mon\text{-}RRWW}(k)) \subseteq \mathsf{CFL} \subseteq \mathcal{L}(\mathsf{mon\text{-}RWW}(k)) \subseteq \mathcal{L}(\mathsf{mon\text{-}RRWW}(k)).$$

Hence, we obtain the following result.

Corollary 3.3. *For all $k \geq 2$,* $\mathsf{CFL} = \mathcal{L}(\mathsf{mon\text{-}RWW}(k)) = \mathcal{L}(\mathsf{mon\text{-}RRWW}(k))$.

4. On deterministic RWW-automata

Here we show that Theorem 3.1 carries over to the deterministic case. The proof consists of two major steps that we present in the following two subsections. In a third subsection, we consider deterministic RWW- and RRWW-automata that are not monotone.

4.1. On deterministic monotone RWW(3)-automata

Let $L \subseteq \Sigma^*$ be a deterministic context-free language. Then there exists a deterministic pushdown automaton (DPDA) $A = (Q_A, \Sigma, \Delta_A, \#, p_0, \delta_A, F_A)$ that accepts L by final state. Here Q_A is a finite set of states, Σ is the

input alphabet, Δ_A is the pushdown alphabet, $\# \notin \Delta_A$ is the bottom marker of the pushdown, $p_0 \in Q_A$ is the initial state,

$$\delta_A : (Q_A \times (\Sigma \cup \{\lambda\}) \times (\Delta_A \cup \{\#\})) \hookrightarrow Q_A \times (\{\#, \lambda\} \cdot \Delta_A^*)$$

is the (partial) transition function, and $F_A \subseteq Q_A$ is the set of final states. We assume without loss of generality that the bottom marker # occurs only as the bottommost symbol on the pushdown and that this symbol is never popped from the pushdown.

A configuration of A is written as $(p, u, \#\alpha)$, where $p \in Q_A$ is the current state, $u \in \Sigma^*$ is the suffix of the input that has not yet been processed, and $\#\alpha \in \# \cdot \Delta_A^*$ is the current contents of the pushdown. Here we assume that the bottom (top) of the pushdown is on the left (right). A word $w \in \Sigma^*$ is accepted by A if the computation of A that starts from the initial configuration $(p_0, w, \#)$ reaches a configuration of the form $(p, \lambda, \#\alpha)$ for some final state $p \in F_A$ and some word $\alpha \in \Delta_A^*$. We use the notation $\vdash_A$ to denote the single-step computation relation that A induces on its set of configurations, and we use $\vdash_A^*$ to denote the corresponding computation relation. Here we establish the following technical result.

Theorem 4.1. *Each DPDA A can be simulated by a deterministic monotone RWW-automaton of window size three.*

Proof. Let $A = (Q_A, \Sigma, \Delta_A, \#, p_0, \delta_A, F_A)$ be a DPDA, and let $L = L(A) \subseteq \Sigma^*$ be the language accepted by A. Without loss of generality we can assume that each λ-step of A pops a symbol from the pushdown, that is, if $\delta_A(p, \lambda, x) = (p', y)$ for some $p, p' \in Q_A$, $x \in \Delta_A \cup \{\#\}$, and $y \in \{\#, \lambda\} \cdot \Delta_A^*$, then $x \in \Delta_A$ and $y = \lambda$ (see, *e.g.*, [1], Prop. 5.4). Thus, given a word $w = a_1 a_2 \cdots a_n \in L(A)$ as input, the computation of A begins with a reading step that reads the first letter a_1, and then there is a sequence of $n-1$ phases that each consists of a (possibly empty) sequence of λ-steps during which some symbols are popped from the pushdown that is followed by a reading step. Finally, after the last reading step, which reads the letter a_n, another sequence of λ-steps that pop some symbols from the pushdown may follow taking A into a final state.

For constructing an RWW-automaton $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, 3, \delta)$ of window size three that simulates the computations of A, we need the constant

$$c_A = \max\{\, |\gamma| \mid \exists\, p, p' \in Q_A, a \in \Sigma, x \in \Delta_A \cup \{\#\} : \delta_A(p, a, x) = (p', \gamma) \,\},$$

that is, c_A is the length of the longest word that A can push onto its pushdown in a single step. Furthermore, let $c = 2c_A - 1$.

The idea of the construction of M is as follows. Using code symbols of the form $[\alpha]$ and $[\alpha, p]$, the contents $\#\alpha_1\alpha_2 \cdots \alpha_m$ of the pushdown of A and the actual state p of A are encoded by a word of the form $[\#\alpha_1][\alpha_2] \cdots [\alpha_m, p]$, where $|\#\alpha_1| = |\alpha_2| = \ldots = |\alpha_{m-1}| = c$ and $1 \leq |\alpha_m| \leq c$. Thus, the contents of the pushdown is compressed by the factor c, and its topmost part is stored together with the current state of A. Processing the input $w = a_1 a_2 \cdots a_n$ from left to right, M tries to simulate a (possibly empty) sequence of λ-steps together with the next reading step. This sequence of steps causes a change in the contents of the pushdown, that is, some symbols may be popped from the pushdown and a new word may be pushed onto the pushdown. If the corresponding change of the pushdown contents can be stored in the rightmost code symbol, then M can simulate these steps by removing the input letter read and by replacing the rightmost code symbol. However, if the rightmost code symbol does not have sufficient storage space left, then an additional code symbol must be used. This causes a problem as each rewrite step of M must be length-reducing. To overcome this problem, M combines the above steps with the next sequence of λ-steps. If these λ-steps decrease the size of the contents of the pushdown sufficiently enough, then the simulation can continue by just manipulating the rightmost code symbol. Finally, if that is not the case, either, then M moves its window one more step to the right and simulates another reading step, reading the next input symbol as well.

To realize this strategy, the RWW(3)-automaton M is now defined by taking

– $Q = \{q_0\}$,
– $\Gamma = \Sigma \cup \{\, [\nu_\#] \mid \nu_\# \in \{\#,\lambda\} \cdot \Delta_A^*, |\nu_\#| = c\}$
$\cup \{\, [\alpha_\#, p] \mid \alpha_\# \in \{\#,\lambda\} \cdot \Delta_A^*, 1 \leq |\alpha_\#| \leq c, p \in Q_A \,\}$,

and the transition function δ is defined by the following table, where $a, b \in \Sigma$, $p, p_1, p_2, p_3, p_4 \in Q_A$, $x, y \in \Delta_A$, $\alpha, \beta, \gamma, \eta, \eta_1, \eta_2, \mu, \nu \in \Delta_A^*$, $\alpha_\#, \nu_\# \in \{\#,\lambda\} \cdot \Delta_A^*$, $1 \leq |\alpha_\#| \leq c$, and $|\nu_\#| = c$:

(1) $\delta(q_0, ¢a\$) = \mathsf{Accept}$ for all $a \in (\Sigma \cup \{\lambda\}) \cap L$,
using this rule M accepts inputs of length at most one,

(2) $\delta(q_0, ¢ab) = ¢[\#\alpha\gamma, p_3]$, if $\delta_A(p_0, a, \#) = (p_1, \#\alpha x\beta)$, $(p_1, \lambda, x\beta) \vdash_A^{|\beta|} (p_2, \lambda, x)$, and $\delta_A(p_2, b, x) = (p_3, \gamma)$,

(3) $\delta(q_0, ¢ab) = ¢[\#\gamma, p_3]$, if $\delta_A(p_0, a, \#) = (p_1, \#\alpha)$, $(p_1, \lambda, \#\alpha) \vdash_A^{|\alpha|} (p_2, \lambda, \#)$, and $\delta_A(p_2, b, \#) = (p_3, \#\gamma)$,

rules (2) and (3) create the first code symbol on the tape,

(4) $\delta(q_0, ¢[\#\alpha x\beta, p]a) = ¢[\#\alpha\gamma, p_2]$, if $(p, \lambda, x\beta) \vdash_A^{|\beta|} (p_1, \lambda, x)$, $\delta_A(p_1, a, x) = (p_2, \gamma)$, and $|\#\alpha\gamma| \leq c$,

(5) $\delta(q_0, ¢[\#\alpha x\beta, p]a) = ¢[\#\alpha\gamma, p_3]$, if $(p, \lambda, x\beta) \vdash_A^{|\beta|} (p_1, \lambda, x)$, $\delta_A(p_1, a, x) = (p_2, \gamma\eta)$, $|\#\alpha\gamma| = c, |\eta| > 0$, and $(p_2, \lambda, \eta) \vdash_A^{|\eta|} (p_3, \lambda, \lambda)$,

(6) $\delta(q_0, ¢[\#\alpha, p]a) = ¢[\#\gamma, p_2]$, if $(p, \lambda, \#\alpha) \vdash_A^{|\alpha|} (p_1, \lambda, \#)$, and $\delta_A(p_1, a, \#) = (p_2, \#\gamma)$,

rules (4)–(6) read the next input symbol if the whole new pushdown contents fits into a single code symbol,

(7) $\delta(q_0, ¢[\#\alpha x\beta, p]a) = (q_0, \mathsf{MVR})$, if $(p, \lambda, x\beta) \vdash_A^{|\beta|} (p_1, \lambda, x)$, $\delta_A(p_1, a, x) = (p_2, \gamma\eta)$, $|\#\alpha\gamma| = c, |\eta| > 0$, but $(p_2, \lambda, \eta) \nvdash_A^{|\eta|}$,

(8) $\delta(q_0, [\alpha_\# x\beta, p]ab) = [\alpha_\#\gamma][\eta_1\mu, p_4]$, if $(p, \lambda, x\beta) \vdash_A^{|\beta|} (p_1, \lambda, x)$, $\delta_A(p_1, a, x) = (p_2, \gamma\eta)$, $|\alpha_\#\gamma| = c, |\eta| > 0, \eta = \eta_1 y \eta_2$, $(p_2, \lambda, y\eta_2) \vdash_A^{|\eta_2|} (p_3, \lambda, y)$, and $\delta_A(p_3, b, y) = (p_4, \mu)$,

rule (8) preceded by (7) or (17) reads the next two input symbols as after reading the first symbol another code symbol is needed,

(9) $\delta(q_0, ¢[\#\alpha][\beta, p]) = (q_0, \mathsf{MVR})$,
(10) $\delta(q_0, ¢[\#\alpha][\beta]) = (q_0, \mathsf{MVR})$,
(11) $\delta(q_0, [\nu_\#][\beta][\gamma, p]) = (q_0, \mathsf{MVR})$,
(12) $\delta(q_0, [\nu_\#][\beta][\gamma]) = (q_0, \mathsf{MVR})$,

rules (9)–(12) move the window over a sequence of code symbols,

(13) $\delta(q_0, [\nu_\#][\alpha, p]a) = [\nu_\#, p_1]a$, if $(p, \lambda, \alpha) \vdash_A^{|\alpha|} (p_1, \lambda, \lambda)$,

rule (13) removes the last code symbol by a sequence of λ-steps,

(14) $\delta(q_0, [\nu_\#][x\beta, p]a) = [\nu_\#, p_2]$, if $(p, \lambda, x\beta) \vdash_A^{|\beta|} (p_1, \lambda, x)$, and $\delta_A(p_1, a, x) = (p_2, \lambda)$,

rule (14) removes the last code symbol by a sequence of λ-steps followed by reading the next input symbol,

(15) $\delta(q_0, [\nu_\#][\alpha x\beta, p]a) = [\nu_\#][\alpha\gamma, p_2]$, if $(p, \lambda, x\beta) \vdash_A^{|\beta|} (p_1, \lambda, x)$, $\delta_A(p_1, a, x) = (p_2, \gamma)$, and $|\alpha\gamma| \leq c$,

rule (15) reads the next input symbol if no additional code symbol is needed,

(16) $\delta(q_0, [\nu_\#][\alpha x\beta, p]a) = [\nu_\#][\alpha\gamma, p_3]$, if $(p, \lambda, x\beta) \vdash_A^{|\beta|} (p_1, \lambda, x)$, $\delta_A(p_1, a, x) = (p_2, \gamma\eta)$, $|\alpha\gamma| = c, |\eta| > 0$, and $(p_2, \lambda, \eta) \vdash_A^{|\eta|} (p_3, \lambda, \lambda)$,

rule (16) reads the next input symbol if no additional code symbol is needed because of a subsequent sequence of λ-steps,

(17) $\delta(q_0, [\nu_\#][\alpha x\beta, p]a) = (q_0, \mathsf{MVR})$, if $(p, \lambda, x\beta) \vdash_A^{|\beta|} (p_1, \lambda, x)$, $\delta_A(p_1, a, x) = (p_2, \gamma\eta)$, $|\alpha\gamma| = c, |\eta| > 0$, but $(p_2, \lambda, \eta) \nvdash_A^{|\eta|} (p_3, \lambda, \lambda)$,

rule (17) realizes a move to the right as reading the next input symbol would require an additional code symbol – this rule can be followed by (8) or (21),

(18) $\delta(q_0, ¢[\#\alpha\beta, p]\$) = \mathsf{Accept}$, if $(p, \lambda, \beta) \vdash_A^{|\beta|} (p_1, \lambda, \lambda)$ and $p_1 \in F_A$,

(19) $\delta(q_0, [\nu_\#][\alpha\beta, p]\$) = \mathsf{Accept}$, if $(p, \lambda, \beta) \vdash_A^{|\beta|} (p_1, \lambda, \lambda)$ and $p_1 \in F_A$,

rules (18) and (19) accept when the whole input has been read,

(20) $\delta(q_0, [\nu_\#][\alpha, p]\$) = [\nu_\#, p_1]\$$, if $(p, \lambda, \alpha) \vdash_A^{|\alpha|} (p_1, \lambda, \lambda)$ without entering a final state,

rule (20) removes the last code symbol by a sequence of λ-steps after the whole input has been read,

(21) $\delta(q_0, [\alpha_\# x\beta, p]a\$) = \mathsf{Accept}$, if $(p, \lambda, x\beta) \vdash_A^{|\beta|} (p_1, \lambda, x)$, $\delta_A(p_1, a, x) = (p_2, \gamma\eta)$, $|\alpha_\#\gamma| = c, |\eta| > 0, \eta = \eta_1 y \eta_2$, $(p_2, \lambda, y\eta_2) \vdash_A^{|\eta_2|} (p_3, \lambda, y)$, and $p_3 \in F_A$,

rule (21) accepts if reading the last input symbol would require an additional code symbol, but a subsequent sequence of λ-steps leads to an accepting state.

As each accepting computation of M obviously simulates an accepting computation of the DPDA A, it follows that $L(M) \subseteq L = L(A)$ holds. On the other hand, we can prove the following claim by induction on the length of the input word w.

Claim. For all $p \in Q_A$, $w \in \Sigma^*$, and $\alpha \in \Delta_A^*$, if $(p, w, \#\alpha) \vdash_A^* (p_f, \lambda, \#\gamma)$ for some $p_f \in F_A$ and $\gamma \in \Delta_A^*$, then

$$q_0 ¢ [\#\alpha_1][\alpha_2] \cdots [\alpha_{m-1}][\alpha_m, p] w \$ \vdash_M^* \mathsf{Accept},$$

where $\alpha = \alpha_1 \alpha_2 \cdots \alpha_m$, $|\#\alpha_1| = |\alpha_2| = \ldots = |\alpha_{m-1}| = c$, and $1 \leq |\alpha_m| \leq c$.

Proof. We proceed by induction on $|w|$. If $w = \lambda$, then the computation $(p, w, \#\alpha) = (p, \lambda, \#\alpha) \vdash_A^* (p_f, \lambda, \#\gamma)$ consists of a (possibly empty) sequence of λ-steps, where $\alpha = \gamma\gamma'$ for some $\gamma' \in \Delta^*$. Thus, $\gamma = \alpha_1 \cdots \alpha_{i-1}\alpha_i'$ and $\gamma' = \alpha_i'' \alpha_{i+1} \cdots \alpha_m$ for some $\alpha_i', \alpha_i'' \in \Delta_A^*$, where $\alpha_i = \alpha_i' \alpha_i''$.

If $|\#\gamma| \geq |\#\alpha_1 \cdots \alpha_{m-1}|$, then M can execute the following computation:

$$\begin{array}{l} q_0 ¢ [\#\alpha_1][\alpha_2] \cdots [\alpha_{m-1}][\alpha_m, p] \$ \\ \qquad \vdash_M^* \quad \mathsf{Accept} \quad \text{(by rule (18) or by rules (9)–(12) and (19)).} \end{array}$$

Otherwise, $|\#\gamma| < |\#\alpha_1 \cdots \alpha_{m-1}|$ and M can execute the following computation:

$$\begin{array}{rll} q_0 ¢ [\#\alpha_1][\alpha_2] \cdots [\alpha_{m-1}][\alpha_m, p] \$ & & \\ = \quad q_0 ¢ [\#\alpha_1] \cdots [\alpha_i' \alpha_i''][\alpha_{i+1}] \cdots [\alpha_m, p] \$ & & \\ \vdash_M^{c^*} \quad q_0 ¢ [\#\alpha_1] \cdots [\alpha_{i-1}][\alpha_i' \alpha_i'', p'] \$ & & \text{(by rules (9)–(12) and (20))} \\ \vdash_M^{c} \quad \mathsf{Accept} & & \text{(by rules (9)–(12), and (18)} \\ & & \text{or (19)),} \end{array}$$

where $p' \in Q_A$ is the state reached by A from the configuration $(p, \lambda, \#\alpha)$ after popping the suffix $\alpha_{i+1} \cdots \alpha_m$ of α.

Now assume that the claim above has been proved for all words w of length up to some $n \geq 0$ and let $w = aw'$ for some letter $a \in \Sigma$ and a word $w' \in \Sigma^n$. Then the accepting computation $(p, w, \#\alpha) \vdash_A^* (p_f, \lambda, \#\gamma)$ of A can be written as follows:

$$(p, w, \#\alpha) = (p, aw', \#\alpha) \vdash_A^* (p_1, aw', \#\omega) \vdash_A (p_2, w', \#\omega') \vdash_A^* (p_f, \lambda, \#\gamma).$$

From the induction hypothesis, we obtain that, for any $p' \in Q_A$ and $\alpha' \in \Delta^*$ such that $(p_2, w', \#\omega') \vdash_A^* (p', w', \#\alpha') \vdash_A^* (p_f, \lambda, \#\gamma)$, the following holds:

$$q_0 ¢ [\#\alpha_1'][\alpha_2'] \cdots [\alpha_{m-1}'][\alpha_m', p'] w' \$ \vdash_M^* \mathsf{Accept},$$

where $\alpha' = \alpha_1' \alpha_2' \cdots \alpha_m'$, $|\#\alpha_1'| = |\alpha_2'| = \ldots = |\alpha_{m-1}'| = c$, and $1 \leq |\alpha_m'| \leq c$.

Furthermore, the above A-computation from the configuration $(p, aw', \#\alpha)$ up to the configuration $(p', w', \#\alpha')$ consists of (at most) three parts: a (possibly empty) sequence of λ-steps, then a step in which the letter a is read, and then possibly another sequence of λ-steps. Thus, it can be written as follows:

(i) either there exist $\alpha', \gamma' \in \Delta^*$ and $p_1, p_2 \in Q_A$ such that $\eta = \alpha'\gamma'$, $\delta_A(p_1, a, \#) = (p_2, \#\eta)$, and

$$(p, aw', \#\alpha) \vdash_A^{|\alpha|} (p_1, aw', \#) \vdash_A (p_2, w', \#\eta) \vdash_A^{|\gamma'|} (p', w', \#\alpha'),$$

(ii) or $\alpha = \beta x \zeta$ for some $\beta, \zeta \in \Delta_A^*$ and $x \in \Delta_A$, $\delta_A(p_1, a, x) = (p_2, \eta)$, and $\beta\eta = \alpha'\gamma'$:

$$(p, aw', \#\alpha) \vdash_A^{|\zeta|} (p_1, aw', \#\beta x) \vdash_A (p_2, w', \#\beta\eta) \vdash_A^{|\gamma'|} (p', w', \#\alpha').$$

In case (i), M can execute the following computation:

$$\begin{array}{ll} & q_0\text{¢}[\#\alpha_1][\alpha_2]\cdots[\alpha_{m-1}][\alpha_m,p]aw'\$ \\ \vdash_M^{c^*} & q_0\text{¢}[\#,p_1]aw'\$ \vdash_M^c q_0\text{¢}[\#\eta,p_2]w'\$ \quad \text{(by rules (9)–(13) and (6))} \\ \vdash_M^* & \mathsf{Accept}, \end{array}$$

where the last part follows from the induction hypothesis for $\alpha' = \eta$ and $p' = p_2$.

In case (ii), M can execute the following computation:

$$\begin{array}{rl} q_0\text{¢}[\#\alpha_1][\alpha_2]\cdots[\alpha_{m-1}][\alpha_m,p]w\$ & \\ = & q_0\text{¢}[\#\beta_1][\beta_2]\cdots[\beta_i x\zeta_1][\zeta_2]\cdots[\zeta_j][\zeta_{j+1},p]aw'\$ \quad (\text{as } \alpha = \beta x\zeta) \\ \vdash_M^{c^*} & q_0\text{¢}[\#\beta_1][\beta_2]\cdots[\beta_{i-1}][\beta_i x\zeta_1,p'']aw'\$ \qquad \text{(by rules (9)–(13))}, \end{array}$$

where $p'' \in Q_A$ is the state that A reaches from the configuration $(p, aw', \#\alpha) = (p, aw', \#\beta_1\cdots\beta_i x\zeta_1\cdots\zeta_{j+1})$ after popping the suffix $\zeta_2\cdots\zeta_{j+1}$ of ζ from its pushdown.

If $|\beta_i\eta| \leq c$, then this computation continues as follows:

$$\begin{array}{rl} q_0\text{¢}[\#\beta_1][\beta_2]\cdots[\beta_{i-1}][\beta_i x\zeta_1,p'']aw'\$ & \\ \vdash_M^c & q_0\text{¢}[\#\beta_1][\beta_2]\cdots[\beta_{i-1}][\beta_i\eta,p_2]w'\$ \quad \text{(by rules (9)–(12) and (4) or (15))} \\ = & q_0\text{¢}[\#\alpha'_1][\alpha'_2]\cdots[\alpha'_{m-1}][\alpha'_m,p']w'\$ \\ \vdash_M^* & \mathsf{Accept}, \end{array}$$

where the last part follows from the induction hypothesis for $\alpha' = \beta\eta$ and $p' = p_2$.

If $|\beta_i\eta| > c$, but $|\#\alpha'| \leq |\#\beta_1\beta_2\cdots\beta_{i-1}| + c$, $\eta = \eta'\eta''$ and $|\beta_i\eta'| = c$, then the above computation continues as follows, where the state p'_2 is reached by A from the configuration $(p_2, w', \#\beta_1\cdots\beta_i x\eta'\eta'')$ by popping η'' from the pushdown:

$$\begin{array}{rl} q_0\text{¢}[\#\beta_1][\beta_2]\cdots[\beta_{i-1}][\beta_i x\zeta_1,p'']aw'\$ & \\ \vdash_M^c & q_0\text{¢}[\#\beta_1][\beta_2]\cdots[\beta_{i-1}][\beta_i\eta',p'_2]w'\$ \quad \text{(by rules (9)–(12), and (5) or (16))} \\ \vdash_M^* & \mathsf{Accept}, \end{array}$$

where the last part follows from the induction hypothesis for $\alpha' = \beta_1\cdots\beta_i\eta'$ and $p' = p'_2$.

Finally, if $|\beta_i\eta| > c$ and $|\#\alpha'| > |\#\beta_1\beta_2\cdots\beta_{i-1}| + c$, then either

(a) $w' = \lambda$, or
(b) $w' \neq \lambda$.

In case (a), we obtain

$$\begin{array}{rll} q_0\text{¢}[\#\beta_1][\beta_2]\cdots[\beta_{i-1}][\beta_i x\zeta_1,p'']aw'\$ & & \\ \vdash_M^* & \mathsf{Accept} & \text{(by rules (9)–(12), and (21))}. \end{array}$$

In case (b), we also have to consider the next part of the above computation of the DPDA A, in which the first letter of w', say b, is read:

$$(p', w', \#\alpha') = (p', bw'', \#\alpha') \vdash_A^* (q, w'', \#\alpha'') \vdash_A^* (p_f, \lambda, \#\gamma).$$

Then the above computation of M continues as follows:

$$\begin{array}{rll} q_0\c[\#\beta_1][\beta_2]\cdots[\beta_{i-1}][\beta_i x\zeta_1, p'']aw'\$ & & \\ = \quad q_0\c[\#\beta_1][\beta_2]\cdots[\beta_{i-1}][\beta_i x\zeta_1, p'']abw''\$ & & \\ \vdash_M^c \quad q_0\c[\#\beta_1][\beta_2]\cdots[\beta_{i-1}][\beta_i\eta_1][\eta_2, q]w''\$ & \text{(by rules (9)–(12),} & \\ & \text{then (7) and (8)} & \\ & \text{or (17) and (8)),} & \end{array}$$

where $\#\beta_1\beta_2\cdots\beta_{i-1}\beta_i\eta_1\eta_2 = \#\alpha''$. Then the induction hypothesis can be applied to the configuration $(q, w'', \#\alpha'')$, which yields

$$q_0\c[\#\beta_1][\beta_2]\cdots[\beta_{i-1}][\beta_i\eta_1][\eta_2, q]w''\$ \vdash_M^* \mathsf{Accept}.$$

□

Let $w \in L(A)$. If $|w| \leq 1$, then M accepts on input w by rule (1). So assume that $|w| \geq 2$, and let $w = abw'$ for some $a, b \in \Sigma$. Then

$$(p_0, abw', \#) \vdash_A (p, bw', \#\alpha) \vdash_A^* (p', w', \#\alpha') \vdash_A^* (p_f, \lambda, \#\gamma)$$

for some $p, p' \in Q_A$, $q_f \in F_A$, and words $\alpha, \alpha', \gamma \in \Delta_A^*$. It is easily seen that $|\#\alpha'| \leq c$, and hence, M can execute the following computation:

$$\begin{array}{rll} q_0\c w\$ \quad = \quad q_0\c abw'\$ & & \\ \vdash_M^c \quad q_0\c[\#\alpha', p']w'\$ & \text{(by rule (2) or (3))} & \\ \vdash_M^* \quad \mathsf{Accept} & \text{(by the claim above).} & \end{array}$$

It follows that $L = L(A) \subseteq L(M)$, and hence, $L(M) = L = L(A)$.

During a computation, the tape contents of M is always of the form

$$\c[\#\alpha_1][\alpha_2]\cdots[\alpha_{m-1}][\alpha_m, p]a_r a_{r+1}\cdots a_n\$,$$

where $\alpha_1, \alpha_2, \ldots, \alpha_m \in \Delta_A^*$, $p \in Q_A$, and $a_r, a_{r+1}, \ldots, a_n \in \Sigma$. As each rewrite step involves the rightmost code symbol on the tape, and as it processes the leftmost input symbol (or the two leftmost input symbols), it follows that M is monotone. Thus, M is a deterministic monotone RWW(3)-automaton for the language $L(A)$. □

From this simulation and from Theorem 2.3 we obtain the following consequence.

Corollary 4.2. $\mathsf{DCFL} = \mathcal{L}(\mathsf{det\text{-}mon\text{-}RWW}(3)) = \mathcal{L}(\mathsf{det\text{-}mon\text{-}RRWW}(3))$.

In addition, we get the following, as for each $k \geq 4$,

$$\mathcal{L}(\mathsf{det\text{-}mon\text{-}RRWW}(k)) \subseteq \mathsf{DCFL} = \mathcal{L}(\mathsf{det\text{-}mon\text{-}RWW}(3)) \subseteq \mathcal{L}(\mathsf{det\text{-}mon\text{-}RWW}(k)).$$

Corollary 4.3. *For all $k \geq 3$,*

$$\mathsf{DCFL} = \mathcal{L}(\mathsf{det\text{-}mon\text{-}RWW}(k)) = \mathcal{L}(\mathsf{det\text{-}mon\text{-}RRWW}(k)).$$

4.2. On deterministic monotone RWW(2)-automata

Here we show that each deterministic monotone RWW-automaton of window size three can be simulated by a deterministic monotone RWW-automaton of only window size two. This simulation will be based on the following important observation.

Proposition 4.4. *Each deterministic RWW- or RRWW-automaton of window size two is monotone.*

Proof. Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, 2, \delta)$ be a deterministic RRWW-automaton of window size two. Each rewrite transition of M replaces a factor of length two of the tape contents by the empty word λ or by a single letter. Thus, if M executes a cycle of the form

$$q_0¢a_1a_2\cdots a_{i-1}a_ia_{i+1}a_{i+2}\cdots a_n\$ \vdash_M^c q_0¢a_1a_2\cdots a_{i-1}a_{i+2}\cdots a_n\$,$$

by deleting the factor a_ia_{i+1}, where $a_1, a_2, \ldots, a_n \in \Gamma$, then this cycle begins with a sequence of i MVR steps that shift the window to the factor a_ia_{i+1} and a subsequent rewrite step that deletes this factor. As M is deterministic, the next cycle begins with a sequence of $i-1$ MVR steps that shift the window to the factor $a_{i-1}a_{i+2}$, which implies that the latter cycle has a smaller right distance than the former.

If M executes a cycle of the form

$$q_0¢a_1a_2\cdots a_{i-1}a_ia_{i+1}a_{i+2}\cdots a_n\$ \vdash_M^c q_0¢a_1a_2\cdots a_{i-1}ba_{i+2}\cdots a_n\$,$$

by replacing the factor a_ia_{i+1} by the letter b, where $a_1, a_2, \ldots, a_n, b \in \Gamma$, then this cycle begins with a sequence of i MVR steps that shift the window to the factor a_ia_{i+1} and the subsequent rewrite step. As M is deterministic, the next cycle begins with a sequence of $i-1$ MVR steps that shift the window to the factor $a_{i-1}b$, which implies that the latter cycle has at most the same right distance as the former.

Together these two cases show that M is indeed monotone. Obviously, the same arguments apply if M is a deterministic RWW-automaton of window size two. □

Thus, we have the following inclusion.

Corollary 4.5. $\mathcal{L}(\mathsf{det\text{-}RWW}(2)) \subseteq \mathcal{L}(\mathsf{det\text{-}RRWW}(2)) \subseteq \mathsf{DCFL}$.

The proof of Proposition 4.4 rests on the fact that, in each cycle, a restarting automaton with a window of size two replaces a factor of length two by a word of length at most one. Thus, this result also extends to deterministic restarting automata of any finite window size $k \geq 3$, provided that each of their rewrite operations rewrites a factor of length k by a word of length at most one. However, if a deterministic restarting automaton of window size $k \geq 3$ contains a rewrite operation of the form $\delta(q, a_1a_2\cdots a_k) = (q', b_1b_2)$, where $a_1, a_2, \ldots, a_k, b_1, b_2$ are letters, then after performing a cycle of the form

$$q_0¢c_1c_2\cdots c_ma_1a_2\cdots a_kd_1d_2\cdots d_n\$ \vdash^c q_0¢c_1c_2\cdots c_mb_1b_2d_1d_2\cdots d_n\$,$$

it could happen that in the next cycle the factor of length k that ends in c_mb_1 is rewritten, which means that the cycle containing this rewrite step has a larger right distance than the previous cycle. Hence, this restarting automaton would not be monotone.

Below we need the following technical result that is based on Lemma 3 of [4] (pages 1806–1808).

Lemma 4.6. *Let A be a deterministic finite-state acceptor with input alphabet Σ. For each word $x = x_1x_2\cdots x_n$, where $x_i \in \Sigma$, and each integer i, $1 \leq i \leq n$, let $q_{x,i}$ denote the state entered by A after processing the prefix $x_1x_2\cdots x_i$. Then there exists a deterministic two-way finite-state acceptor A' such that, for each input $x = x_1x_2\cdots x_n$ and each i, $2 \leq i \leq n$, if A' starts its computation on x in state $q_{x,i}$ with its head on x_i, then A' finishes its computation in state $q_{x,i-1}$ with its head on the symbol x_{i-1}. Moreover, during this computation A' only visits (a part of) $x_1x_2\cdots x_i$.*

Now we come to the main result of this subsection.

Theorem 4.7. *Each deterministic monotone RWW-automaton of window size three can be simulated by a deterministic RWW-automaton of window size two.*

Proof. Using Lemma 4.6 we can show the following. Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, 3, \delta)$ be a deterministic monotone RWW-automaton, and let $F(M)$ be the set of all words $x \in \Gamma^*$ such that, after starting from the restarting configuration $q_0¢x\$$, M will accept eventually, but the first rewrite/restart step within this computation is performed at the right end of the tape (that is, with the right sentinel \$ in its read/write window). Then $F(M)$ is a regular language.

In fact, if $x \in F(M)$, then all rewrite/restart steps within the computation of M that starts from the configuration $q_0¢x\$$ occur at the right sentinel \$, as M is monotone. In addition, we can assume without loss of generality that the accept step is also performed at the right sentinel. Furthermore, observe that the move-right steps of M can be simulated by a deterministic finite-state acceptor A_{MVR}. Based on Lemma 4.6, we obtain a deterministic two-way finite-state acceptor A'_{MVR} that can simulate the reverse transitions of A_{MVR}.

Now we construct a two-way finite-state acceptor A_1 for the set $F(M)$. The acceptor A_1 simulates the RWW-automaton M while scanning the tape from left to right. If M is to execute a rewrite/restart step before reaching the right sentinel, then A_1 halts in a non-final state. When M reaches the right end of its tape (that is, when the right sentinel \$ appears in the read/write window), then, instead of executing the rewrite/restart step of M, the acceptor A_1 simulates the rewrite step within its finite-state control and

(a) it remembers the contents of the rewritten tape between the leftmost letter in the window of M and the right sentinel, and
(b) by using the deterministic two-way finite-state acceptor A'_{MVR}, it computes the state that M enters when it reaches the first symbol that was produced by the rewrite step within the previous cycle. Then it continues the simulation of the computation of M from this point onwards. When M halts accepting or rejecting, then A_1 accepts or rejects as well. Otherwise, A_1 simulates the next rewrite step in its finite-state control and continues.

Obviously, the computation of A_1 is finite for any input word x. Thus, it follows that $F(M)$ is a regular language. In addition, from A_1, we can construct a (one-way) finite-state acceptor $A = (Q_A, \Gamma, \delta_A, q_{A,0}, F_A)$ for the language $F(M)$.

We now show how to construct a deterministic RWW-automaton $M' = (Q', \Sigma, \Gamma', ¢, \$, q'_0, 2, \delta')$ of window size two that accepts the same language as M.

The read/write window of M' is smaller than the read/write window of M. Hence, in order to simulate a cycle of M, the automaton M' may have to perform more than one cycle. Accordingly, its state will be used to represent not only the current state of M, but also the symbol immediately to the left of its current window position and the corresponding state of the finite-state acceptor A. Hence, the set of states of M' will be

$$Q' = \{\, [q, x, p] \mid q \in Q, x \in \Gamma \cup \{¢, \lambda\}, p \in Q_A \,\}.$$

The initial state of M' is $q'_0 = [q_0, \lambda, q_{A,0}]$, and the first step performed by M' from a restarting configuration with a word x on its tape is

$$[q_0, \lambda, q_{A,0}]¢x\$ \vdash_{M'} ¢[q_0, ¢, q_{A,0}]x\$.$$

A series of move-right steps of M can be simulated easily. For each move-right step $\delta(q, u_1u_2u_3) = (q', \mathsf{MVR})$ of M, the automaton M' has the move-right step

$$\delta'([q, u_1, p], u_2u_3) = ([q', u_2, p'], \mathsf{MVR})$$

for all states $p, p' \in Q_A$ such that $\delta_A(p, u_2) = p'$.

However, the simulation of the rewrite steps of M is more complicated. First of all, no rewrite at the right end of the tape of M needs to be simulated, as all the remaining rewrite steps by M must then also occur at the right end of the tape. Instead, M' can inspect the state of A after scanning the last letter of the current contents of the tape and accept or reject immediately. Thus, M' will only simulate a rewrite of M if the corresponding contents of the window of M does not contain the right sentinel \$. Any combined rewrite/restart step of M according to

$$\delta(q, u_1u_2u_3) = v, \text{ where } u_1, u_2, u_3 \in \Gamma \text{ and } v \in \Gamma^{\leq 2}, \tag{4.1}$$

will be simulated as follows:

(i) If $v = \lambda$, the automaton M' rewrites u_2u_3 by a special symbol $\Delta \notin \Gamma$ and restarts. When in a subsequent cycle the symbol Δ appears in the read/write window of M', the automaton deletes Δ and the symbol immediately to the left of Δ. This completes the simulation of (4.1) by M'.

(ii) If $v = v_1$ for some $v_1 \in \Gamma$, the automaton M' rewrites u_2u_3 by the pair $[\Delta, v_1]$ and restarts. When $[\Delta, v_1]$ appears as the second symbol within the read/write window of M', the automaton rewrites the contents of its read/write window by v_1, in this way completing the simulation of (4.1).

(iii) If $v = v_1v_2$ for some $v_1, v_2 \in \Gamma$, the automaton M' cannot simulate this rewrite step in two cycles, as two cycles would shorten the tape by at least two symbols. The automaton M' must first read the tape until it sees u_2u_3 (preceded by u_1). If $v_1 = u_1$, the automaton simply rewrites u_2u_3 into v_2 and the simulation of (4.1) is complete. Otherwise, M' rewrites u_2u_3 into the symbol $[v_1, v_2]$ with the meaning that the contents of the corresponding tape field is v_2 and its left neighbour should be v_1. Because of the monotonicity of M, after performing the combined rewrite/restart step using (4.1), M cannot perform another rewrite within the next cycle before its window contains the whole factor $v = v_1v_2$ (otherwise the right distance of the latter cycle would be larger than that of the former). Therefore, the simulating automaton M' will scan the tape until it sees $u_1[v_1, v_2]$. It can then interpret the contents of both fields containing u_1 and $[v_1, v_2]$ properly as fields containing v_1 and v_2. Of course, the transition relation δ' of M' must also include move-right instructions over fields containing pairs of symbols, as the next rewrite after using (4.1) can occur to the right of v_1, and in addition, all rewrite instructions of M' must preserve the information within the first element of a pair symbol on its tape.

In general, the tape of M' will contain symbols from Γ and pairs of symbols from $\Gamma \times \Gamma$:

- The automaton M' interprets a symbol $x \in \Gamma$ on its tape as x on the tape of M, if it is not followed by a pair, and as v_1 on the tape of M, if it is followed by a pair $[v_1, v_2]$.
- The automaton M' interprets a symbol $[x, y] \in \Gamma \times \Gamma$ on its tape as y on the tape of M, if it is not followed by a pair, and as v_1 on the tape of M, if it is followed by a pair $[v_1, v_2]$.

As M is deterministic and monotone, all pair symbols on the tape of M' will be scanned by M' prior to simulating any further rewrites by M. Hence, all symbols on the tape of M' will be interpreted correctly and the automaton M' is deterministic. Moreover, as all computations of deterministic RWW(2)-automata are monotone by Proposition 4.4, it follows that M' is monotone. □

Because of Theorem 4.1 the above simulation has the following consequence.

Corollary 4.8. $\mathsf{DCFL} = \mathcal{L}(\mathsf{det\text{-}mon\text{-}RWW(2)}) = \mathcal{L}(\mathsf{det\text{-}mon\text{-}RRWW(2)})$
$= \mathcal{L}(\mathsf{det\text{-}RWW(2)}) = \mathcal{L}(\mathsf{det\text{-}RRWW(2)})$.

In combination, our results yield the following construction of a deterministic monotone RWW-automaton of window size two from a deterministic monotone RWW-automaton of window size $k \geq 4$. Let M be a deterministic monotone RWW-automaton of window size $k \geq 4$. From M, we can first construct a DPDA P for the language $L(M)$ as shown in the proof of Theorem 2.2 in [7]. From this DPDA, we can derive a deterministic monotone

RWW-automaton of window size three for $L(M)$ as in the proof of Theorem 4.1, and then, using the construction above, we obtain a deterministic (monotone) RWW-automaton of window size two for $L(M)$.

4.3. On deterministic RWW-automata

Here we turn to deterministic RWW-automata that are not (necessarily) monotone. We have seen above that deterministic RWW- and RRWW-automata of window size two are necessarily monotone, which implies that they yield a characterization for the class DCFL of deterministic context-free languages. What can be said about deterministic RWW- and RRWW-automata of window size larger than two?

We begin with a simple example, considering the non-context-free language

$$L_{\text{expo}} = \{\, a^{2^n} \mid n \geq 0 \,\}.$$

It is known that L_{expo} is a Church-Rosser language [17], and hence, it is accepted by a deterministic RWW-automaton [24]. In fact, it is even accepted by a deterministic RWW-automaton of window size three.

Example 4.9. Let $M = (Q, \{a\}, \Gamma, ¢, \$, q_0, 3, \delta)$ be the deterministic RWW-automaton of window size three that is defined by taking $Q = \{q_0\}$, $\Gamma = \{a, b\}$, and by defining the transition function δ as follows:

$$\begin{array}{llcl@{\qquad}llcl}
(1) & \delta(q_0, ¢a\$) & = & \mathsf{Accept}, & (6) & \delta(q_0, ¢b\$) & = & \mathsf{Accept},\\
(2) & \delta(q_0, ¢aa) & = & (q_0, \mathsf{MVR}), & (7) & \delta(q_0, ¢bb) & = & (q_0, \mathsf{MVR}),\\
(3) & \delta(q_0, aaa) & = & (q_0, \mathsf{MVR}), & (8) & \delta(q_0, bbb) & = & (q_0, \mathsf{MVR}),\\
(4) & \delta(q_0, aa\$) & = & b\$, & (9) & \delta(q_0, bb\$) & = & a\$,\\
(5) & \delta(q_0, aab) & = & bb, & (10) & \delta(q_0, bba) & = & aa.
\end{array}$$

It is easily seen that M accepts the language L_{expo}. Also observe that M is non-monotone, as, for example,

$$q_0¢aaaaaaaa\$ \vdash_M^c q_0¢aaaaaab\$ \vdash_M^c q_0¢aaaabb\$ \vdash_M^c q_0¢aabbb\$ \vdash_M^{c^*} \mathsf{Accept}$$

is a non-monotone accepting computation.

As $L_{\text{expo}} \in \mathcal{L}(\mathsf{det\text{-}RWW}(3))$, we have the following chain of inclusions, where CRL denotes the class of Church-Rosser languages:

$$\begin{array}{lcl@{\quad}lcl}
\mathsf{REG} & = & \mathcal{L}(\mathsf{det\text{-}R(R)WW}(1)) & & & \\
& \subsetneq & \mathcal{L}(\mathsf{det\text{-}R(R)WW}(2)) & = & \mathsf{DCFL} & \\
& \subsetneq & \mathcal{L}(\mathsf{det\text{-}RWW}(3)) & & & \\
& \subseteq & \mathcal{L}(\mathsf{det\text{-}RRWW}(3)) & \subseteq & \mathcal{L}(\mathsf{det\text{-}R(R)WW}) & = \quad \mathsf{CRL}.
\end{array}$$

It remains open whether the latter inclusions are proper or not.

5. On shrinking restarting automata

We now turn to a generalized type of restarting automaton, the shrinking restarting automaton, which was introduced in [26] and further studied in [9–11]. A *shrinking restarting automaton* M is defined just like an RRWW-automaton with the one exception that it is no longer required that each rewrite step $u \to v$ of M must be length-reducing. Instead, there must exist a weight function ω that assigns a positive integer $\omega(a)$ to each letter a of M's tape alphabet Γ such that, for each rewrite step $u \to v$ of M, $\omega(u) > \omega(v)$. Here the function ω is extended to a morphism $\omega : \Gamma^* \to \mathbb{N}$ by taking $\omega(\lambda) = 0$ and $\omega(wa) = \omega(w) + \omega(a)$ for all words $w \in \Gamma^*$ and letters $a \in \Gamma$. We use the notation sRRWW and sRWW to denote shrinking RRWW- and RWW-automata.

In [10], the following result has been established, where FCA denotes the class of finite-change automata from [30]. A *finite-change automaton* is a nondeterministic linear-bounded automaton that does not change the contents of any tape cell more than r times during any accepting computation, where $r \geq 1$ is a constant.

Theorem 5.1. [10] $\mathcal{L}(\mathsf{FCA}) = \mathcal{L}(\mathsf{sRWW}) = \mathcal{L}(\mathsf{sRRWW})$.

In the proof of Lemma 17 in [10], it is shown that a finite-change automaton A can be simulated by an sRRWW-automaton M with window size one. In fact, the sRRWW(1)-automaton M just performs rewrite steps that replace a single letter by another single letter. Hence, M can be interpreted as an ordered RRWW-automaton (ORRWW-automaton) as defined in [14]. These are sRRWW-automata that have a window of size three, but the rewrite steps of which just replace the symbol in the middle of the window by a smaller letter, that is, by a letter of less weight. This yields the following consequence, which answers the question about the expressive power of ORRWW-automata that was left open in [14].

Corollary 5.2. $\mathcal{L}(\mathsf{FCA}) = \mathcal{L}(\mathsf{sRRWW}(1)) = \mathcal{L}(\mathsf{ORRWW})$.

The simulation of an sRRWW-automaton of window size k by an sRWW-automaton given in [10] yields that the latter has window size $\max\{2k, 9\}$. Thus, we obtain the following result.

Corollary 5.3. $\mathcal{L}(\mathsf{FCA}) = \mathcal{L}(\mathsf{sRWW}(9))$.

However, it remains open whether sRWW-automata of window size $k \in \{2, 3, \ldots, 8\}$ are really less expressive than finite-change automata. Concerning sRWW(1)-automata, we have the following results.

Proposition 5.4. $\mathsf{REG} \subsetneq \mathcal{L}(\mathsf{sRWW}(1))$.

Proof. Obviously, a finite-state acceptor can be simulated by an sRWW(1)-automaton that does not use any rewrite/restart operations. Thus, $\mathsf{REG} \subseteq \mathcal{L}(\mathsf{sRWW}(1))$.

In order to prove that this inclusion is proper, we consider the example language

$$L_{\geq} = \{\, a^{m+n}b^n \mid m, n \geq 0 \,\},$$

which is easily seen to be a deterministic context-free language that is not regular. Now an sRWW(1)-automaton $M_{\geq} = (Q, \{a, b\}, \Gamma, ¢, \$, q_0, 1, \delta)$ for the language $L_{\geq}$ can be defined by taking $Q = \{q_0, q_1, q_2, p_0, p_1, p_2\}$, $\Gamma = \{a, b, a_1, a_2, a_3, b_1, b_2\}$, and by defining the transition relation δ as follows:

$$
\begin{array}{llcl}
(1) & \delta(q_0, ¢) & = & \{(q_0, \mathsf{MVR})\}, \\
(2) & \delta(q_0, \$) & = & \{\mathsf{Accept}\}, \\
(3) & \delta(q_0, a) & = & \{a_1\}, \\
(4) & \delta(q_0, a_1) & = & \{(q_1, \mathsf{MVR}), a_2\}, \\
(5) & \delta(q_0, a_2) & = & \{(q_2, \mathsf{MVR}), a_3\}, \\
(6) & \delta(q_0, a_3) & = & \{(q_0, \mathsf{MVR})\}, \\
(7) & \delta(q_0, b_2) & = & \{(p_0, \mathsf{MVR})\}, \\
(8) & \delta(p_0, b_2) & = & \{(p_0, \mathsf{MVR})\}, \\
(9) & \delta(p_0, \$) & = & \{\mathsf{Accept}\}, \\
(10) & \delta(q_1, a) & = & \{(q_1, \mathsf{MVR})\}, \\
(11) & \delta(q_1, b) & = & \{b_1\},
\end{array}
\qquad
\begin{array}{llcl}
(12) & \delta(q_1, b_2) & = & \{(p_1, \mathsf{MVR})\}, \\
(13) & \delta(q_1, \$) & = & \{\mathsf{Accept}\}, \\
(14) & \delta(q_2, a) & = & \{(q_2, \mathsf{MVR})\}, \\
(15) & \delta(q_2, b_1) & = & \{b_2\}, \\
(16) & \delta(q_2, b_2) & = & \{(p_2, \mathsf{MVR})\}, \\
(17) & \delta(p_1, b_2) & = & \{(p_1, \mathsf{MVR})\}, \\
(18) & \delta(p_1, b) & = & \{b_1\}, \\
(19) & \delta(p_1, \$) & = & \{\mathsf{Accept}\}, \\
(20) & \delta(p_2, b_2) & = & \{(p_2, \mathsf{MVR})\}, \\
(21) & \delta(p_2, b_1) & = & \{b_2\}.
\end{array}
$$

From the form of the transitions, it is easily seen that $M_{\geq}$ performs an accepting tail computation iff the tape contents is of the form $z = a_3^m b_2^n$ for some $m, n \geq 0$ or of the form $a_3^m a_1 a^r b_2^n$ for some $m, n, r \geq 0$. Thus, the corresponding input is of the form $a^s b^t$ for some $s, t \geq 0$. During an accepting computation on an input of this form, the occurrences of the letter a are rewritten, from left to right, first into a_1, then into a_2, and finally into a_3. Analogously, the occurrences of the letter b are rewritten, from left to right, first into b_1 and then into b_2. However, an occurrence of b can only be rewritten into b_1 (by (11) or (18)) if in that cycle, the

rightmost occurrence of the letter a that has already been rewritten happens to be an a_1 (see (4) and (12)), and an occurrence of b_1 can only be rewritten into b_2 (by (15) or (21)) if in that cycle, the rightmost occurrence of the letter a that has already been rewritten happens to be an a_2 (see (5) and (16)). It follows that $s \geq t$, that is, $L(M_{\geq}) = L_{\geq}$. Thus, REG is indeed a proper subclass of $\mathcal{L}(\mathsf{sRWW}(1))$. □

The technique used in the construction of the sRWW(1)-automaton for the language $L_{\geq}$ in the above proof can easily be extended to show that also the language

$$L'_{\text{copy}} = \{\, w\#u \mid w, u \in \{a,b\}^*, |w|, |u| \geq 2, u \text{ is a scattered subword of } w \,\}$$

is accepted by an sRWW(1)-automaton. In [15], it has been shown that this language is not even growing context-sensitive.

Proposition 5.4 establishes a lower bound for the expressive power of sRWW(1)-automata. Next we derive an upper bound for it. Recall from [15, 16] that an ORWW-automaton is an sRWW(3)-automaton the rewrite steps of which just replace the symbol in the middle of the window by a smaller letter, that is, by a letter with less weight.

Proposition 5.5. $\mathcal{L}(\mathsf{sRWW}(1)) \subseteq \mathcal{L}(\mathsf{ORWW})$.

Proof. Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, 1, \delta)$ be a shrinking RWW(1)-automaton that is compatible with the weight function $\varphi : \Gamma \to \mathbb{N}_+$, and let $c = \max\{\, \varphi(a) \mid a \in \Sigma \,\}$, that is, c is the maximal weight of an input letter. Without loss of generality, we can assume that M only accepts with its read/write window on the right sentinel \$.

We construct an ORWW-automaton $M_o = (Q, \Sigma, \Delta, ¢, \$, q_0, 3, \delta_o, >)$, where $>$ is a partial ordering on the tape alphabet Δ such that $x > y$ holds whenever M_o rewrites a symbol x into y. First we take

$$\Delta = \Sigma \cup \{\, [\alpha] \mid \alpha \in \Gamma^*, \varphi(\alpha) \leq c \,\}.$$

Here the symbol $[\alpha]$ encodes the word $\alpha \in \Gamma^*$, where the weight $\varphi(\alpha)$ is bounded from above by the constant c. As the weight of each letter $a \in \Gamma$ is positive, we see that there are only finitely many words from Γ^* with weight bounded from above by c. Hence, Δ is indeed a finite alphabet. We define a morphism $\psi : \Delta^* \to \Gamma^*$ through $\psi(a) = a$ for all $a \in \Sigma$ and $\psi([\alpha]) = \alpha$ for all $[\alpha] \in \Delta \setminus \Sigma$, and we define the partial ordering $>$ on Δ by taking, for all $a, b \in \Sigma$, $a > [a]$, and for all $x, y \in \Delta \setminus \Sigma$,

$$x > y \text{ if } \varphi(\psi(x)) > \varphi(\psi(y)).$$

Finally, we specify the transition relation δ_o as follows, where $\vdash_{\text{MVR}}$ denotes the single-step computation relation that is induced by the MVR steps of M, $a, b \in \Sigma$, $u, v, z, \alpha \in \Gamma^*$, $\omega \in \Delta$, $X \in \Gamma$, and $q_1, q_2 \in Q$:

$$\begin{array}{lllll}
(1) & \delta_o(q_0, ¢\$) & = & \{\mathsf{Accept}\}, & \text{if } q_0¢\$ \vdash_{\text{MVR}} ¢q_1\$ \\
 & & & & \text{and } \delta(q_1, \$) = \{\mathsf{Accept}\}, \\
(2) & \delta_o(q_0, ¢a\$) & = & \{¢[a]\$\}, & \\
(3) & \delta_o(q_0, ¢[u]\$) & = & \{\mathsf{Accept}\}, & \text{if } q_0¢u\$ \vdash_{\text{MVR}}^{|u|+1} ¢uq_1\$ \\
 & & & & \text{and } \delta(q_1, \$) = \{\mathsf{Accept}\}, \\
(4) & \delta_o(q_0, ¢[uXv]\$) & \ni & ¢[uzv]\$, & \text{if } q_0¢uXv\$ \vdash_{\text{MVR}}^{|u|+1} ¢uq_1Xv\$ \\
 & & & & \text{and } \delta(q_1, X) \ni z, \\
(5) & \delta_o(q_0, ¢ab) & = & \{¢[a]b\}, & \\
(6) & \delta_o(q_0, ¢[u]\omega) & \ni & (q_1, \mathsf{MVR}), & \text{if } q_0¢u\psi(\omega) \vdash_{\text{MVR}}^{|u|+1} ¢uq_1\psi(\omega),
\end{array}$$

$$
\begin{array}{lllll}
(7) & \delta_o(q_0, ¢[uXv]\omega) & \ni & ¢[uzv]\omega, & \text{if } q_0¢uXv \vdash_{\mathrm{MVR}}^{|u|+1} ¢uq_1Xv \\
 & & & & \text{and } \delta(q_1, X) \ni z, \\
(8) & \delta_o(q_1, [\alpha]ab) & = & \{[\alpha][a]b\}, & \\
(9) & \delta_o(q_1, [\alpha][u]\omega) & \ni & (q_2, \mathsf{MVR}), & \text{if } q_1u\psi(\omega) \vdash_{\mathrm{MVR}}^{|u|} uq_2\psi(\omega), \\
(10) & \delta_o(q_1, [\alpha][uXv]\omega) & \ni & [\alpha][uzv]\omega, & \text{if } q_1uXv \vdash_{\mathrm{MVR}}^{|u|} uq_2Xv \\
 & & & & \text{and } \delta(q_2, X) \ni z, \\
(11) & \delta_o(q_1, [\alpha]a\$) & = & \{[\alpha][a]\$\}, & \\
(12) & \delta_o(q_1, [\alpha][u]\$) & = & \{\mathsf{Accept}\}, & \text{if } q_1u\$ \vdash_{\mathrm{MVR}}^{|u|} uq_2\$ \\
 & & & & \text{and } \delta(q_2, \$) = \{\mathsf{Accept}\}, \\
(13) & \delta_o(q_1, [\alpha][uXv]\$) & \ni & [\alpha][uzv]\$, & \text{if } q_1uXv \vdash_{\mathrm{MVR}}^{|u|} uq_2Xv \\
 & & & & \text{and } \delta(q_2, X) \ni z.
\end{array}
$$

Note that instruction (9) also simulates the process of moving over a deleted part of the tape of M, as $\delta_o(q_1, [\alpha][\lambda]\omega) \ni (q_1, \mathsf{MVR})$ for all states $q_1 \in Q$. All rewrite steps of M_o do either replace an input letter a by the auxiliary letter $[a]$ or they simulate a rewrite step of M on some auxiliary symbol. Hence, each rewrite step replaces the letter in the middle of the window by a smaller one with respect to the partial ordering $>$. Thus, we see that M_o is indeed an ORWW-automaton.

It remains to prove that M_o accepts the same (input) language as M. It is easily seen from the definition of δ_o that M_o accepts on input λ iff M accepts on input λ. So let $w = a_1a_2 \cdots a_n$ be given as input, where $n \geq 1$ and $a_1, a_2, \ldots, a_n \in \Sigma$. Assume that $w \in L(M)$, that is, M has an accepting computation on input w. As M only accepts with its read/write window on the right sentinel \$, this computation looks as follows:

$$
\begin{array}{lllllll}
q_0¢w\$ & = & q_0¢a_1a_2 \cdots a_n\$ & \vdash_M^c & q_0¢w_1\$ & \vdash_M^c & q_0¢w_2\$ \\
 & \vdash_M^c & \cdots & \vdash_M^c & q_0¢w_m\$ & \vdash_{\mathrm{MVR}}^+ & ¢w_mq_+\$ \\
 & \vdash_M & \mathsf{Accept}, & & & &
\end{array}
$$

where $q_+ \in Q$, $\delta(q_+, \$) = \{\mathsf{Accept}\}$, and $w_1, w_2, \ldots, w_m \in \Gamma^*$. As M is shrinking with window size one, each word w_i can be factored as $w_i = u_{i,1}u_{i,2} \cdots u_{i,n}$, where $u_{i,j}$ is the factor of w_i that is obtained from the input letter a_i by those rewrite steps in the above computation that have been applied to a_i and to the i-th factors of $w_2, w_3, \ldots, w_{i-1}$. Note that the length of $u_{i,j}$ can be greater than one as an sRWW(1)-automaton can rewrite a single symbol x into a word y, provided that $\varphi(x) > \varphi(y)$.

Now the ORWW-automaton M_o proceeds as follows. On input $w = a_1a_2 \ldots a_n$, it replaces each letter a_i by the auxiliary symbol $[a_i]$ using instructions (2), (5), (8), and (11). Then it simulates the cycles of M by performing the rewrite steps from M's computation within the corresponding auxiliary symbols, that is, if in a cycle

$$
\begin{array}{lll}
q_0¢w_i\$ & = & q_0¢u_{i,1}u_{i,2} \cdots u_{i,j-1}u_{i,j}u_{i,j+1} \cdots u_{i,n}\$ \\
 & \vdash_M^c & q_0¢u_{i,1}u_{i,2} \cdots u_{i,j-1}u_{i+1,j}u_{i,j+1} \cdots u_{i,n}\$ \\
 & = & q_0¢w_{i+1}\$
\end{array}
$$

the factor $u_{i,j}$ is rewritten into the factor $u_{i+1,j}$, then M_o executes the cycle

$$
\begin{array}{lll}
q_0¢w_i'\$ & = & q_0¢[u_{i,1}][u_{i,2}] \cdots [u_{i,j-1}][u_{i,j}][u_{i,j+1}] \cdots [u_{i,n}]\$ \\
 & \vdash_M^c & q_0¢[u_{i,1}][u_{i,2}] \cdots [u_{i,j-1}][u_{i+1,j}][u_{i,j+1}] \cdots [u_{i,n}]\$ \\
 & = & q_0¢w_{i+1}'\$
\end{array}
$$

by rewriting the letter $[u_{i,j}]$ into the letter $[u_{i+1,j}]$ using instruction (4), (7), (10), or (13). Here it is possible that some of the factors $u_{i,j+1}$ to $u_{i,n}$ are just input letters that have not yet been rewritten into auxiliary

symbols by M_o. Finally, as

$$q_0 ¢ w_m \$ = q_0 ¢ u_{m,1} u_{m,2} \cdots u_{m,n} \$ \vdash^*_{\mathrm{MVR}} ¢ u_{m,1} u_{m,2} \cdots u_{m,n} q_+ \$$$

and $\delta(q_+, \$) = \{\mathsf{Accept}\}$, M_o can execute an accepting tail computation starting from the configuration

$$q_0 ¢ w'_m \$ = q_0 ¢ [u_{m,1}][u_{m,2}] \cdots [u_{m,n}] \$$$

using instructions (3), (6), (9), and (12).

Conversely, it is easily seen that each accepting computation of M_o is just the simulation of an accepting computation of M. Thus, $L(M_0) = L(M)$ follows, which completes the proof of Proposition 5.5. □

Next we turn to shrinking RWW-automata of window size two. We shall prove that window size two is sufficient for these automata to enable them to accept all growing context-sensitive languages. A language $L \subseteq \Sigma^*$ is *growing context-sensitive* if it is generated by a grammar $G = (N, \Sigma, S, P)$ such that, for each production $(\ell \to r) \in P$, $\ell = S$ or $|\ell| < |r|$ (see, *e.g.*, [3]). Concerning the class GCSL of growing context-sensitive languages, the following characterization has been established, where sTPDA denotes the class of shrinking two-pushdown automata (see below).

Theorem 5.6. [2] $\mathsf{GCSL} = \mathcal{L}(\mathsf{sTPDA})$.

The two-pushdown automaton is defined as follows, where this definition is taken from [2].

Definition 5.7. A *two-pushdown automaton (TPDA)* is a nondeterministic automaton with two pushdown stores. Formally, it is defined as a 7-tuple $A = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$, where

- Q is the finite set of states,
- Σ is the finite input alphabet,
- Γ is the finite tape alphabet with $\Gamma \supsetneq \Sigma$ and $\Gamma \cap Q = \emptyset$,
- $q_0 \in Q$ is the initial state,
- $\bot \in \Gamma \setminus \Sigma$ is the bottom marker of the pushdown stores,
- $F \subseteq Q$ is the set of final (or halting) states, and
- $\delta\colon Q \times \Gamma \times \Gamma \to \mathcal{P}_{\mathrm{fin}}(Q \times \Gamma^* \times \Gamma^*)$ is the transition relation, where $\mathcal{P}_{\mathrm{fin}}(Q \times \Gamma^* \times \Gamma^*)$ denotes the set of finite subsets of $Q \times \Gamma^* \times \Gamma^*$.

Given a word $w \in \Sigma^*$ as input, the TPDA A starts in its initial state q_0 with its first pushdown only containing the bottom marker $\bot$ and its second pushdown containing the input w followed by the bottom marker. Thus, at the start, A sees the bottom marker on its first pushdown and the first letter of w on its second pushdown.

The automaton A is a *deterministic two-pushdown automaton (DTPDA)*, if δ is a (partial) function from $Q \times \Gamma \times \Gamma$ into $Q \times \Gamma^* \times \Gamma^*$.

A configuration of the (D)TPDA A is described as uqv, where $q \in Q$ is the actual state, $u \in \Gamma^*$ is the contents of the first pushdown store with the first letter of u at the bottom and the last letter of u at the top, and $v \in \Gamma^*$ is the contents of the second pushdown store with the last letter of v at the bottom and the first letter of v at the top. Thus, for an input string $w \in \Sigma^*$, the corresponding initial configuration is $\bot q_0 w \bot$. By applying a transition $(q, u, v) \in \delta(p, a, b)$, the automaton replaces the symbol a on the top of its first pushdown store by the word u and the symbol b on the top of its second pushdown store by the word v, and it changes from state p to state q. The (D)TPDA A induces a computation relation $\vdash^*_A$ on the set of configurations, which is the reflexive transitive closure of the single-step computation relation $\vdash_A$ (see, *e.g.*, [5]). The (D)TPDA A accepts with empty pushdown stores, that is,

$$L(A) := \{\, w \in \Sigma^* \mid \bot q_0 w \bot \ \vdash^*_A q \text{ for some } q \in F \,\}$$

is the *language accepted by* A.

Definition 5.8. A (D)TPDA is called *shrinking* if there exists a weight function $\varphi : Q \cup \Gamma \to \mathbb{N}_+$ such that, for all $q \in Q$ and $a, b \in \Gamma$, $(p, u, v) \in \delta(q, a, b)$ implies that $\varphi(upv) < \varphi(aqb)$. By s(D)TPDA we denote the class of shrinking (deterministic) two-pushdown automata.

Thus, if A is a shrinking TPDA with weight-function φ, then $\varphi(u_1q_1v_1) > \varphi(u_2q_2v_2)$ holds for all configurations $u_1q_1v_1$ and $u_2q_2v_2$ of A that satisfy $u_1q_1v_1 \vdash_A u_2q_2v_2$. Observe that the input is provided to a TPDA as the initial contents of its second pushdown store, and that in order to accept a TPDA is required to empty its pushdown stores. Thus, it is forced to consume its input completely. Using standard techniques from automata theory, it can be shown that, for a (shrinking) (deterministic) TPDA $A = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$, we may require that the special symbol $\bot$ can only occur at the bottom of a pushdown store, and that no other letter can occur at that place.

From the definition of the transition relation δ, we see that A halts immediately whenever one of its pushdown stores is emptied. Because of the above property, this happens if and only if a transition of the form $(q, a, \bot) \mapsto (q', u, \lambda)$ or $(q, \bot, b) \mapsto (q', \lambda, v)$ is performed. Thus, we can assume without loss of generality that, if A does accept on input $w \in \Sigma^*$, then $\bot q_0 w \bot \vdash_A^* q$ for some $q \in F$, and if A does not accept on input $w \in \Sigma^*$, then $\bot q_0 w \bot \vdash_A^* \bot q$ for some $q \in F$, that is, even in this situation, A empties its second pushdown store completely and only leaves the bottom marker on its first pushdown store before it halts. Hence, all the halting and accepting configurations of A are of the form q, where $q \in F$, and all the halting and rejecting configurations of A are of the form $\bot q$, where $q \in F$. In addition, we can assume that A has a single halting state only.

Based on Theorem 5.6 and the fact that $\mathsf{GCSL} \subsetneq \mathcal{L}(\mathsf{sRWW}) = \mathcal{L}(\mathsf{FCA})$ (see, *e.g.*, [8]), we can now derive the following inclusion result.

Theorem 5.9. $\mathsf{GCSL} \subsetneq \mathcal{L}(\mathsf{sRWW}(2))$.

Proof. As the language L'_{copy} mentioned above is accepted by an sRWW(1)-automaton, we see that $\mathcal{L}(\mathsf{sRWW}(2))$ contains a language that is not growing context-sensitive. So it remains to prove that each growing context-sensitive language is accepted by some sRWW(2)-automaton.

Let $L \subseteq \Sigma^*$ be a growing context-sensitive language. By Theorem 5.6, there exists an sTPDA $A = (P, \Sigma, \Gamma, \delta, p_0, \bot, \{p_f\})$ that accepts the language L. Let $\varphi : P \cup \Gamma \to \mathbb{N}_+$ be a weight function such that A is shrinking with respect to this weight function. Without loss of generality we can assume that $\varphi(aqb) - \varphi(upv) \geq 2$ for each transition $(p, u, v) \in \delta(q, a, b)$. We now present an sRWW(2)-automaton M that simulates the computations of A.

First, we choose $\Sigma' = \{\, a' \mid a \in \Sigma \,\}$ as a new input alphabet that is disjoint from Γ, but that is in one-to-one correspondence to Σ. Now let $M = (Q, \Sigma', \Omega, ¢, \$, q_0, 2, \delta_M)$ be defined by taking $Q = \{q_0\}$, $\Omega = \Sigma' \cup P \cup \Gamma \cup (\Gamma \times P)$, and by defining the transition relation δ_M as follows, where $p, p_1 \in P$, $a \in \Sigma$, $x, y, z \in \Gamma$, and $u, v \in \Gamma^*$:

$$
\begin{array}{lllll}
(1) & \delta_M(q_0, ¢\$) & = & \{\mathsf{Accept}\}, & \text{if } \delta(p_0, \bot, \bot) = \{(p_f, \lambda, \lambda)\},\\
(2) & \delta_M(q_0, ¢a') & \ni & ¢[\bot, p_0]a, & \\
(3) & \delta_M(q_0, ¢\bot) & = & \{(q_0, \mathsf{MVR})\}, & \\
(4) & \delta_M(q_0, ¢[\bot, p]) & = & \{(q_0, \mathsf{MVR})\}, & \\
(5) & \delta_M(q_0, [\bot, p]\$) & = & \{\mathsf{Accept}\}, & \text{if } (p_f, \lambda, \lambda) \in \delta(p, \bot, \bot),\\
(6) & \delta_M(q_0, xy) & = & \{(q_0, \mathsf{MVR})\}, & \\
(7) & \delta_M(q_0, x[y, p]) & = & \{(q_0, \mathsf{MVR})\}, & \\
(8) & \delta_M(q_0, xp) & = & \{[x, p]\}, & \\
(9) & \delta_M(q_0, [x, p]a') & = & \{[x, p]a\}, & \\
(10) & \delta_M(q_0, [x, p]\$) & \ni & u[z, p_1]v\$, & \text{if } (p_1, uz, v\bot) \in \delta(p, x, \bot),\\
(11) & \delta_M(q_0, [x, p]\$) & \ni & p_1v\$, & \text{if } (p_1, \lambda, v\bot) \in \delta(p, x, \bot),\\
(12) & \delta_M(q_0, [x, p]y) & \ni & u[z, p_1]v, & \text{if } (p_1, uz, v) \in \delta(p, x, y),\\
(13) & \delta_M(q_0, [x, p]y) & \ni & p_1v, & \text{if } (p_1, \lambda, v) \in \delta(p, x, y).
\end{array}
$$

Furthermore, we define a weight function $\omega : \Omega \to \mathbb{N}_+$ as follows:

- $\omega(a') = \varphi(a) + \varphi(\bot) + \varphi(p_0) + 1$ for all $a \in \Sigma$,
- $\omega(x) = \varphi(x)$ for all $x \in \Gamma$,
- $\omega(p) = \varphi(p) + 1$ for all $p \in P$,
- $\omega([x,p]) = \varphi(x) + \varphi(p)$ for all $x \in \Gamma$ and $p \in P$.

Using this weight function, it is easily verified that the RWW-automaton M is shrinking. Indeed, instructions of types (2) and (9) are weight-reducing because of the weight associated to the letters from Σ', instructions of type (8) are weight-reducing because of the weight associated to the letters from the subalphabet P of Ω, instructions of types (10) and (12) are weight-reducing, as the corresponding instructions of A are weight-reducing with respect to φ, and the instructions of types (11) and (13) are weight-reducing because of our assumption that each instruction of A reduces the weight of the corresponding configuration by at least two.

It remains to show that $L(M) = h(L(A))$, where $h : \Sigma^* \to \Sigma'^*$ is the morphism induced by mapping a to a' for all $a \in \Sigma$. A configuration of the form $\bot uxpv\bot$ of A, where $u, v \in \Gamma^*$, $x \in \Gamma$, and $p \in P$, is encoded by the tape contents $¢\bot u[x,p]v\$$ of M, or by the tape contents $¢\bot uxpv\$$. The latter type of encoding is used whenever a step of A is being simulated that only pops from the lefthand pushdown (see instructions (11) and (13)). However, as soon as such an encoding is detected, it is immediately transformed into an encoding of the former form by instruction (8). Now using this encoding of the configurations of A, the sRWW-automaton M simulates the computations of A step by step. Hence, it follows that $L(M) = h(L(A))$.

By renaming the letters of Ω accordingly, an sRWW(2)-automaton is obtained that accepts the language $L(A)$. This completes the proof of Theorem 5.9. □

It is easily seen that each ORWW-automaton can be simulated by an sRWW(2)-automaton. Based on a pumping lemma for ORWW-automata, it has been shown in [16] that the deterministic linear language $L = \{ a^n b^n \mid n \geq 0 \}$ is not accepted by any ORWW-automaton. As this language is accepted by an sRWW(2)-automaton, we have the following chain of inclusions.

Corollary 5.10. $\mathsf{REG} \subsetneq \mathcal{L}(\mathsf{sRWW}(1)) \subseteq \mathcal{L}(\mathsf{ORWW}) \subsetneq \mathcal{L}(\mathsf{sRWW}(2)) \subseteq \mathcal{L}(\mathsf{sRWW}(9)) = \mathcal{L}(\mathsf{FCA})$.

It currently remains open whether the inclusions $\mathcal{L}(\mathsf{sRWW}(1)) \subseteq \mathcal{L}(\mathsf{ORWW})$ and $\mathcal{L}(\mathsf{sRWW}(2)) \subseteq \mathcal{L}(\mathsf{sRWW}(9))$ are proper.

Concerning monotone sRWW- and sRRWW-automata of window size one and two, we now derive the following results.

Theorem 5.11. (a) $\mathcal{L}(\mathsf{mon\text{-}sRWW}(1)) = \mathsf{REG} \subsetneq \mathcal{L}(\mathsf{mon\text{-}sRRWW}(1))$.
(b) $\mathcal{L}(\mathsf{mon\text{-}sRWW}(2)) = \mathsf{CFL} = \mathcal{L}(\mathsf{mon\text{-}sRRWW}(2))$.

Proof. (a) Obviously, each regular language is accepted by some monotone sRWW(1)-automaton that does not execute any rewrite step. On the other hand, the non-regular language

$$L_2' = \{ a^m b^n \mid m \in \{n, n+1\}, n \geq 0 \}$$

is accepted by a monotone RRWW(1)-automaton, and so it is contained in $\mathcal{L}(\mathsf{mon\text{-}sRRWW}(1))$. Indeed, let $M = (Q, \{a,b\}, \{a,b\}, ¢, \$, q_0, 1, \delta)$ be the RRWW(1)-automaton that is defined by taking $Q = \{q_0, q_1, q_2, p_0, p_1, p_2\}$ and by defining the transition relation δ as follows:

(1)	$\delta(q_0, ¢)$	$=$	$\{(q_0, \mathsf{MVR})\}$,	(7)	$\delta(q_1, b)$	$=$	$\{(p_1, \lambda)\}$,
(2)	$\delta(q_0, \$)$	$=$	$\{\mathsf{Accept}\}$,	(8)	$\delta(q_2, b)$	$=$	$\{(p_2, \lambda)\}$,
(3)	$\delta(q_0, a)$	$=$	$\{(p_0, \lambda), (q_2, \mathsf{MVR})\}$,	(9)	$\delta(p_1, b)$	$=$	$\{(p_2, \mathsf{MVR})\}$,
(4)	$\delta(q_1, a)$	$=$	$\{(p_2, \lambda), (q_2, \mathsf{MVR})\}$,	(10)	$\delta(p_2, b)$	$=$	$\{(p_1, \mathsf{MVR})\}$,
(5)	$\delta(q_2, a)$	$=$	$\{(p_1, \lambda), (q_1, \mathsf{MVR})\}$,	(11)	$\delta(p_2, \$)$	$=$	$\{\mathsf{Restart}\}$.
(6)	$\delta(p_0, \$)$	$=$	$\{\mathsf{Accept}\}$,				

Obviously, M accepts the empty word λ and the word a. Furthermore, it is easily seen that M only accepts words from the regular language $a^* \cdot b^*$. Now let $w = a^m b^n$ be given as input, that is, we have the initial configuration $q_0 ¢ a^m b^n \$$. If M deletes one of the first $m-1$ occurrences of the letter a, then it gets stuck immediately thereafter. Hence, we see that M either deletes the last occurrence of the letter a or the first occurrence of the letter b, which implies that M is monotone. From the indices of the states used, we see that to complete a cycle through the restart operation (11), M must delete the last occurrence of the letter a, if m and n do not have the same parity mod 2, and M must delete the first occurrence of the letter b, if m and n have the same parity mod 2. Indeed, depending on the parities of m and n, we have the following cycles:

$$
\begin{array}{lllll}
(o,e) & q_0¢a^{2r+1}b^{2s}\$ & \vdash_M^{2r+1} & ¢a^{2r}q_1ab^{2s}\$ & \vdash_M\ ¢a^{2r}p_2b^{2s}\$ \\
 & & \vdash_M^{2s} & ¢a^{2r}b^{2s}p_2\$ & \vdash_M\ q_0¢a^{2r}b^{2s}\$, \\
(e,o) & q_0¢a^{2r}b^{2s+1}\$ & \vdash_M^{2r} & ¢a^{2r-1}q_2ab^{2s+1}\$ & \vdash_M\ ¢a^{2r-1}p_1b^{2s+1}\$ \\
 & & \vdash_M^{2s+1} & ¢a^{2r-1}b^{2s+1}p_2\$ & \vdash_M\ q_0¢a^{2r-1}b^{2s+1}\$, \\
(e,e) & q_0¢a^{2r}b^{2s}\$ & \vdash_M^{2r+1} & ¢a^{2r}q_1b^{2s}\$ & \vdash_M\ ¢a^{2r}p_1b^{2s-1}\$ \\
 & & \vdash_M^{2s-1} & ¢a^{2r}b^{2s-1}p_2\$ & \vdash_M\ q_0¢a^{2r}b^{2s-1}\$, \\
(o,o) & q_0¢a^{2r+1}b^{2s+1}\$ & \vdash_M^{2r+2} & ¢a^{2r+1}q_2b^{2s+1}\$ & \vdash_M\ ¢a^{2r+1}p_2b^{2s}\$ \\
 & & \vdash_M^{2s} & ¢a^{2r+1}b^{2s}p_2\$ & \vdash_M\ q_0¢a^{2r+1}b^{2s}\$.
\end{array}
$$

Furthermore, in each of these cases, if another occurrence of the letter b is added, in this way changing the relative parities of a's and b's, then M reaches the right sentinel while being in state p_1, and hence, it gets stuck. Thus, it follows that $L(M) = L_2'$.

In Lemma 5.13, we show that each monotone sRWW(1)-automaton can be simulated by a stateless ORWW-automaton. As $\mathcal{L}(\mathsf{stl\text{-}ORWW}) = \mathsf{REG}$ according to [15], this yields (a).

(b) As the simulation of a monotone RRWW-automaton by a pushdown automaton used in [7] to show that monotone RRWW-automata only accept context-free languages easily extends to monotone sRRWW-automata, it follows that $\mathcal{L}(\mathsf{mon\text{-}sRWW}) = \mathcal{L}(\mathsf{mon\text{-}sRRWW}) = \mathsf{CFL}$. This fact, together with Theorem 3.1, yields (b), as each monotone RWW(2)-automaton is also a monotone sRWW(2)-automaton. □

Remark 5.12. Let us extend the RRWW(1)-automaton M from the proof of Theorem 5.11 (a) into a shrinking RRWW(1)-automaton M' as follows: during the first cycle, M' replaces the first occurrence of the letter a on its tape by an auxiliary letter a' and it verifies that the number of a's has the same parity mod 2 as the number of b's. In the affirmative, it then simulates M step by step, interpreting the letter a' in the same way as M would interpret the letter a, while in the negative, it just halts without accepting. Then M' is a monotone sRRWW(1)-automaton for the language $L_2 = \{\, a^m b^m \mid m \geq 0 \,\}$, which is obviously not accepted by any RRWW(1)-automaton. In fact, by counting the number of a's mod 2 and the number of b's mod 4, a monotone sRRWW(1)-automaton can be designed for the language $L = \{\, a^m b^m c, a^m b^{2m} d \mid m \geq 0, \}$, which is not even deterministic context-free. However, it currently remains open whether there exists a context-free language that is not accepted by any monotone sRRWW(1)-automaton. We conjecture that $L_{\mathrm{pal}} = \{\, ww^R \mid w \in \{a,b\}^* \,\}$ is such a language, but currently we do not yet have an idea of how to prove that this language cannot be accepted by any monotone sRRWW(1)-automaton.

Lemma 5.13. *For each monotone sRWW(1)-automaton M, there exists a stateless ORWW-automaton M_s such that $L(M_s) = L(M)$.*

Proof. Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, 1, \delta)$ be a monotone sRWW(1)-automaton that is compatible with the weight function $\varphi : \Gamma \to \mathbb{N}_+$. Here we can assume without loss of generality that M accepts only at the right end of the tape, that is, when it sees the right sentinel \$.

In the proof of Proposition 5.5, we have seen that each sRWW(1)-automaton can be simulated by an ORWW-automaton. That simulation is based on the observation that, by using an enlarged tape alphabet, each rewrite

step of an sRWW(1)-automaton can be interpreted as replacing a letter by either a letter with a smaller weight or by the empty word λ. Thus, we can assume without loss of generality that M already has this property.

Next we study the implications that the monotonicity has on the computations of M. Let

$$C_1 : q_0¢uavbw\$ \vdash_M^{|u|+1} ¢uq_1avbw\$ \vdash_M^{|v|+1} ¢uavq_2bw\$ \vdash_M q_0¢uavyw\$$$

be a cycle of M, where $y \in \delta(q_2, b)$, and assume that $x \in \delta(q_1, a)$. Thus, in the above cycle, M moves across the distinguished letter a by executing a MVR step $(p_1, \mathsf{MVR}) \in \delta(q_1, a)$, and it completes the cycle by the rewrite/restart step $y \in \delta(q_2, b)$, but it could have executed the rewrite/restart step $x \in \delta(q_1, a)$ instead. Hence, the cycle above, which has the right distance $D_r(C_1) = |w| + 2$, can be followed by the cycle

$$C_2 : q_0¢uavyw\$ \vdash_M^{|u|+1} ¢uq_1avyw\$ \vdash_M q_0¢uxvyw\$,$$

which has the right distance $D_r(C_2) = |v| + |y| + |w| + 2$. As M is monotone, this means that $D_r(C_2) = |v| + |y| + |w| + 2 \leq D_r(C_1) = |w| + 2$, which implies that $|v| = 0 = |y|$, that is, $v = y = \lambda$. Thus, the cycle C_1 actually looks as follows:

$$C_1 : q_0¢uabw\$ \vdash_M^{|u|+1} ¢uq_1abw\$ \vdash_M ¢uaq_2bw\$ \vdash_N q_0¢uaw\$,$$

where $\lambda \in \delta(q_2, b)$.

We now define a stateless ORWW-automaton $M_s = (\Sigma, \Delta, ¢, \$, 3, \delta_s, >)$. To simplify the notation, the set of states is omitted from the definition and the states are omitted from the transition relation δ_s. We take

$$\Delta = \Sigma \cup \{\, [i, x, P] \mid 0 \leq i \leq |\Gamma|, x \in \Gamma \cup \{\lambda\}, P \subseteq Q \,\},$$

define the partial ordering $>$ on Δ through

$$\begin{array}{lccll} - & a & > & B & \text{for all } a \in \Sigma \text{ and } B \in \Delta \setminus \Sigma, \\ - & [i, x, P] & > & [j, y, P'] & \text{for all } i < j \text{ and all } x, y \in \Gamma \cup \{\lambda\}, P, P' \subseteq Q, \end{array}$$

and define the transition relation δ_s as follows, where $a, b, c \in \Sigma$, $i, j \in \{0, 1, \ldots, |\Gamma|\}$, $P, P' \subseteq Q$, $x, y, z \in \Gamma$, $x_\lambda \in \Gamma \cup \{\lambda\}$, and $X \in \Delta$:

$$\begin{array}{lllll} (1) & \delta_s(¢a\$) & = & \{\mathsf{Accept}\} \text{ for all } a \in (\Sigma \cup \{\lambda\}) \cap L(M), \\ (2) & \delta_s(¢ab) & \ni & ¢[0, a, P_a]b, \text{ where } P_a = \{\, p \in Q \mid q_0¢a \vdash_M^2 ¢ap \,\}, \\ (3) & \delta_s(¢ab) & \ni & ¢[1, x, P_x]b, \\ & & & \text{where } P_x = \{\, p \in Q \mid q_0¢a \vdash_M^2 q_0¢x \vdash_M^2 ¢xp \,\}, \\ (4) & \delta_s(¢ab) & \ni & ¢[1, \lambda, P]b, \\ & & & \text{where } P = \{\, p \in Q \mid q_0¢a \vdash_M^2 q_0¢ \vdash_M ¢p \,\}, \\ (5) & \delta_s(¢[i, x, P]X) & = & \{\mathsf{MVR}\} \cup \\ & & & \{\, ¢[i+1, y, P_y]X \mid q_0¢x \vdash_M ¢qx \vdash_M q_0¢y \,\} \cup \\ & & & \{\, ¢[i+1, \lambda, P']X \mid i \geq 1 \text{ and } q_0¢x \vdash_M^2 q_0¢ \,\}, \\ & & & \text{where } P_y = \{\, p \in Q \mid q_0¢y \vdash_M^2 ¢yp \,\} \\ & & & \text{and } P' = \{q \in Q \mid q_0¢ \vdash_M ¢q \,\}, \\ (6) & \delta_s(¢[i, \lambda, P]X) & = & \{\mathsf{MVR}\}, \end{array}$$

$$
\begin{array}{lll}
(7) & \delta_s([i,x_\lambda,P]bc) & \ni [i,x_\lambda,P][0,b,P_b]c, \\
 & & \quad \text{where } P_b = \{\, q \in Q \mid \exists p \in P : (q,\mathsf{MVR}) \in \delta(p,b) \,\}, \\
(8) & \delta_s([i,x_\lambda,P]bc) & \supseteq \{\, [i,x_\lambda,P][1,y,P_y]c \mid \exists p \in P : y \in \delta(p,b) \,\}, \text{ where} \\
 & & \quad P_y = \{\, q \in Q \mid \exists p' \in P : (q,\mathsf{MVR}) \in \delta(p',y) \,\}, \\
(9) & \delta_s([i,x_\lambda,P]bc) & \ni [i,x_\lambda,P][i,\lambda,P]c, \text{ if } \exists p \in P : \lambda \in \delta(p,b), \\
(10) & \delta_s([i,x_\lambda,P][j,z,P']X) = \{\mathsf{MVR}\} \cup & \\
 & \quad \{\, [i,x_\lambda,P][j+1,y,P_y]X \mid \exists p \in P : y \in \delta(p,z) \,\} \cup & \\
 & \quad \{[i,x_\lambda,P][j+1,\lambda,P]X \mid j \geq 1 \text{ and } \exists p \in P : \lambda \in \delta(p,z) \,\}, & \\
 & \quad \text{where } P_y = \{\, q \in Q \mid \exists p' \in P : (q,\mathsf{MVR}) \in \delta(p',y) \,\}, & \\
(11) & \delta_s([i+1,x_\lambda,P][i,\lambda,P']X) \ni [i+1,x_\lambda,P][i+1,\lambda,P]X, & \\
(12) & \delta_s(X[i,x_\lambda,P]\$) = \{\mathsf{Accept}\}, \text{ if } \exists p \in P : \mathsf{Accept} \in \delta(p,\$). &
\end{array}
$$

Here a letter of the form $[i,x,P]$ encodes the information that, at the current tape position, M currently has the letter x, that this letter has already been rewritten i times, and that by scanning the current tape contents from left to right, M can reach any of the states in P after having read the letter x. A letter of the form $[i,\lambda,P]$ encodes the information that the letter at the current tape position has been rewritten to λ, and that this was done either by the i-th rewrite at the current position or by the first rewrite at the current position, in which case i is the number of rewrites that have been applied so far at the first position to the left of the current position that has not yet been rewritten to λ. Finally, P is again the set of states that M can reach by scanning the current tape contents from left to right up to the current position.

It remains to prove that $L(M_s) = L(M)$. Instructions (2) and (7) encode state information of M into the letters that are moved over by M's read/write window through MVR steps. Instructions (3), (5), (8), and (10) simulate rewrite steps of M that rewrite a letter by another letter and they encode the corresponding state information in the newly written symbols. Furthermore, instructions (4), (9), and (10) simulate rewrite steps of M that replace a letter by the empty word, and instruction (11) updates state information that is stored in symbols of the form $[i,\lambda,P]$ that has become outdated because a rewrite step was performed somewhere to the left of this symbol. Observe that the λ-rewrite steps of the form (10) (with $x_\lambda = \lambda$) can only be applied to letters that have already been rewritten before, which means that later no rewrite steps can be performed to the left of these letters due to the monotonicity of M. From this description it is easily seen that all accepting computations of M_s are simulations of accepting computations of M, that is, we have $L(M_s) \subseteq L(M)$.

To prove the converse inclusion, let $w = a_1 a_2 \cdots a_n$ be a word from $L(M)$, where $n \geq 0$ and $a_1, a_2, \ldots, a_n \in \Sigma$. If $n \leq 1$, then $w \in (\Sigma \cup \{\lambda\}) \cap L(M)$, and so M_s can accept on input w by using instruction (1). Assume that $n \geq 2$ and let

$$q_0 ¢ w \$ = q_0 ¢ w_0 \$ \vdash_M^c q_0 ¢ w_1 \$ \vdash_M^c \cdots \vdash_M^c q_0 ¢ w_m \$ \vdash_M^* q_0 ¢ w_m q_+ \$$$

be an accepting computation of M on input w, where $q_+ \in Q$ and $\mathsf{Accept} \in \delta(q_+,\$)$. As M is monotone, the above computation is monotone. If all rewrites are performed strictly from left to right, then it is easily seen that M_s can simulate this computation, that is, M_s will also accept on input w.

Finally, assume that

$$w_i = b_1 b_2 \cdots b_{j-1} \underline{b_j} b_{j+1} \cdots b_r \underline{b_{r+1}} b_{r+2} \cdots b_t,$$

that w_{i+1} is obtained from w_i by rewriting the letter b_{r+1} into c, that w_{i+2} is obtained from w_{i+1} by rewriting the letter b_j into d, and that this is the first such situation in the above computation. Then we see from our discussion above that $r = j$ and that $c = \lambda$, that is,

$$
\begin{array}{lll}
w_i & = & b_1 b_2 \cdots b_{j-1} \underline{b_j b_{r+1}} b_{r+2} \cdots b_t, \\
w_{i+1} & = & b_1 b_2 \cdots b_{j-1} \underline{b_j} b_{r+2} \cdots b_t, \text{ and} \\
w_{i+2} & = & b_1 b_2 \cdots b_{j-1} \underline{d b_{r+2}} \cdots b_t.
\end{array}
$$

In particular, none of the letters to the left of d can be rewritten anymore. By simulating the steps of the above computation of M up to w_i, M_s obtains the restarting configuration

$$\underline{¢[i_1, b_1, P_1][i_2, b_2, P_2]}[i_3, b_3, P_3] \cdots [i_j, b_j, P_j] b_{r+1} b_{r+2} \cdots b_t\$,$$

where we underline the symbols inside the read/write window of M_s. Now by instruction (5), (6), and (10), M_s can move its window to the right until it reaches the configuration

$$¢[i_1, b_1, P_1][i_2, b_2, P_2] \cdots [i_{j-1}, b_{j-1}, P_{j-1}]\underline{[i_j, b_j, P_j] b_{r+1} b_{r+2}} b_{r+3} \cdots b_t\$.$$

As P_j is the set of all states that M can reach by reading the prefix $¢b_1 b_2 \cdots b_j$ of the tape contents from left to right, there exists a state $q \in P_j$ such that $\lambda \in \delta(q, b_{r+1})$, and hence, M_s can rewrite the letter b_{r+1} into $[i_j, \lambda, P_j]$ by instruction (9), which yields the configuration

$$\underline{¢[i_1, b_1, P_1][i_2, b_2, P_2]}[i_3, b_3, P_3] \cdots [i_j, b_j, P_j][i_j, \lambda, P_j] b_{r+2} \cdots b_t\$.$$

In the next cycle $[i_j, b_j, P_j]$ is rewritten into $[i_j + 1, d, P']$, and then $[i_j, \lambda, P_j]$ can be rewritten into $[i_j + 1, \lambda, P']$ by instruction (11), which yields the configuration

$$\underline{¢[i_1, b_1, P_1][i_2, b_2, P_2]}[i_3, b_3, P_3] \cdots [i_j + 1, d, P'][i_j + 1, \lambda, P'] b_{r+2} \cdots b_t\$.$$

It now follows that M_s can simulate the above computation of M, implying that $w \in L(M_s)$. Hence, it follows that $L(M_s) = L(M)$.

This completes the proof of Lemma 5.13 and, therewith, of Theorem 5.11. □

6. On deterministic shrinking restarting automata

Finally, we consider deterministic sRWW- and sRRWW-automata. The construction in the proof of Proposition 5.5 carries over to the deterministic case, and it can easily be extended to simulate a det-sRRWW(1)-automaton by a det-ORRWW-automaton. As det-ORWW-automata and det-ORRWW-automata only accept regular languages (see [14, 19]), we obtain the following results.

Theorem 6.1. (a) $\mathcal{L}(\text{det-sRWW(1)}) = \mathcal{L}(\text{det-ORWW}) = \text{REG}$.
(b) $\mathcal{L}(\text{det-sRRWW(1)}) = \mathcal{L}(\text{det-ORRWW}) = \text{REG}$.

Also the simulation of an sTPDA by an sRWW(2)-automaton in the proof of Theorem 5.9 carries over to the deterministic case. As deterministic sTPDAs characterize the class CRL of Church-Rosser languages [25], this yields the following result.

Corollary 6.2.

$$\mathcal{L}(\text{det-sRWW(2)}) = \mathcal{L}(\text{det-sRRWW(2)}) = \mathcal{L}(\text{det-sRWW}) = \mathcal{L}(\text{det-sRRWW}) = \text{CRL}.$$

Thus, we see that for deterministic shrinking RWW- and RRWW-automata, the hierarchy based on window size consists of only two levels: window size one yields the regular languages, and window size $k \geq 2$ yields the Church-Rosser languages.

Concerning monotone deterministic sRWW- and sRRWW-automata of window size one or two, the above results yield the following consequences.

Corollary 6.3.
(a) $\mathcal{L}(\text{det-mon-sRWW(1)}) = \mathcal{L}(\text{det-mon-sRRWW(1)}) = \text{REG}$.
(b) $\mathcal{L}(\text{det-mon-sRWW(2)}) = \mathcal{L}(\text{det-mon-sRRWW(2)}) = \text{DCFL}$.

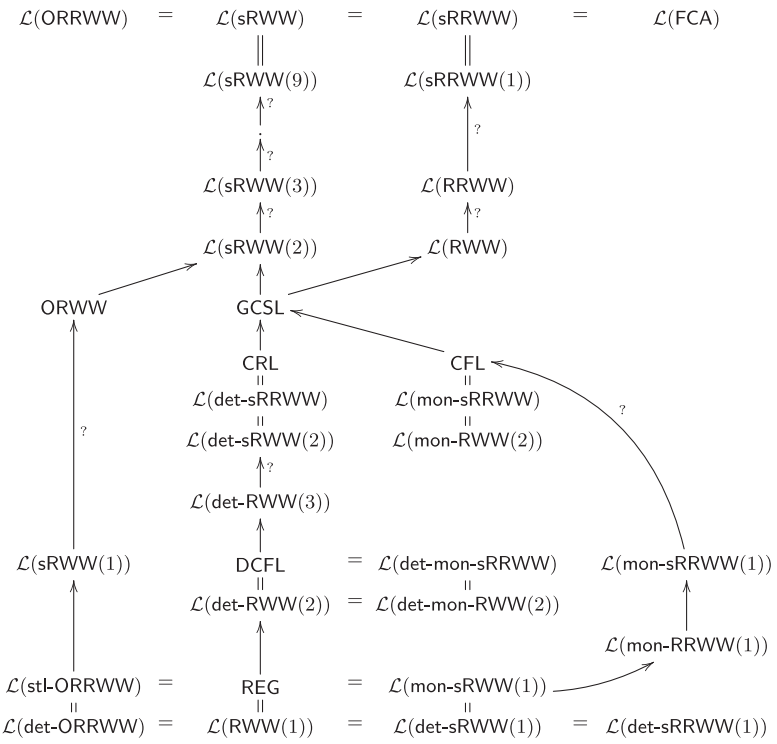

FIGURE 1. The taxonomy of (shrinking) restarting automata with small window size. An arrow $\rightarrow$ denotes a proper inclusion, while $\rightarrow^{?}$ denotes an inclusion that is not known to be proper.

7. Conclusion

We have studied the expressive power of restarting automata and shrinking restarting automata of small window size. We have seen that for deterministic as well as for nondeterministic RWW- and RRWW-automata that are monotone, window size two suffices to accept all languages that can be accepted by these types of automata. In fact, our proofs provide constructions that allow converting a (deterministic) monotone RWW-automaton of window size $k \geq 3$ into an equivalent (deterministic) monotone RWW-automaton of window size two.

It is known that deterministic RWW- and RRWW-automata of window size one just accept the regular languages [13, 18]. Here we have seen that deterministic RWW- and RRWW-automata of window size two are necessarily monotone, which implies that they yield a characterization for the class DCFL of deterministic context-free languages. What can be said about deterministic RWW- and RRWW-automata of window size larger than two? Here we have the following chain of inclusions

$$\begin{array}{ccccc} \mathsf{DCFL} & \subsetneq & \mathcal{L}(\mathsf{det\text{-}RWW}(3)) & \subseteq & \mathcal{L}(\mathsf{det\text{-}RRWW}(3)) \\ & \subseteq & \mathcal{L}(\mathsf{det\text{-}R(R)WW}) & = & \mathsf{CRL}, \end{array}$$

but it remains open whether the latter inclusions are proper. In fact, it is open whether there exists an integer $k \geq 3$ such that $\mathcal{L}(\mathsf{det\text{-}RWW}(k)) = \mathcal{L}(\mathsf{det\text{-}RWW}) = \mathsf{CRL}$, or whether the language classes $(\mathcal{L}(\mathsf{det\text{-}RWW}(k))_{k\geq 3}$ (and $(\mathcal{L}(\mathsf{det\text{-}RRWW}(k))_{k\geq 3})$ yield an infinite ascending hierarchy within the class CRL of Church-Rosser languages. Also it is open whether $\mathcal{L}(\mathsf{det\text{-}RWW}(k)) = \mathcal{L}(\mathsf{det\text{-}RRWW}(k))$ for any $k \geq 3$. Recall that in [29], Natalie Schluter has only shown that $\mathcal{L}(\mathsf{det\text{-}RRWW}(k)) \subseteq \mathcal{L}(\mathsf{det\text{-}RWW}(2k-2))$ for all $k \geq 2$. Thus, we see that for deterministic

RWW- and RRWW-automata, the influence of the window size on the expressive power of these automata is still unsolved.

Concerning shrinking restarting automata, we have seen that for sRRWW-automata, already window size one suffices, while for monotone and/or deterministic sRWW- and sRRWW-automata, window size two is required to obtain the full expressive power of these types of automata. In particular, we have seen that for deterministic shrinking RWW- and RRWW-automata, the hierarchy based on window size consists of only two levels: window size one yields the regular languages, and window size $k \geq 2$ yields the Church-Rosser languages. Also it remains open whether sRWW(1)-automata are as powerful as ORWW-automata, whether window size nine is really needed to obtain the full power of sRWW-automata, and whether monotone sRRWW(1)-automata accept all context-free languages. The diagram in Figure 1 summarizes the characterizations and inclusion relations we have obtained for (shrinking) restarting automata of small window size.

Recall from Remark 5.12 that the language $L = \{ a^m b^m c, a^m b^{2m} d \mid m \geq 0 \}$ is accepted by a monotone sRRWW(1)-automaton. This language is context-free, but not deterministic context-free. In fact, L is a linear language that even belongs to the class 2-detLIN of languages that are accepted by deterministic two-head finite automata that read their tape from both ends [22]. Accordingly, it may be of interest to study the relationship between the class 2-detLIN and the various language classes specified by shrinking restarting automata of window size one.

References

[1] J.-M. Autebert, J. Berstel and L. Boasson, Context-free languages and pushdown automata, in *Handbook of Formal Languages, Vol. 1, Word, Language, Grammar*, edited by G. Rozenberg and A. Salomaa. Springer, Berlin (1997) 111–174.

[2] G. Buntrock and F. Otto, Growing context-sensitive languages and Church-Rosser languages. *Inf. Comput.* **141** (1998) 1–36.

[3] E. Dahlhaus and M. Warmuth, Membership for growing context-sensitive grammars is polynomial. *J. Comput. Syst. Sci.* **33** (1986) 456–472.

[4] J.E. Hopcroft and J.D. Ullman, An approach to a unified theory of automata. *Bell Syst. Tech. J.* **46** (1967) 1793–1829.

[5] J.E. Hopcroft and J.D. Ullman, Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, MA (1979).

[6] P. Jančar, F. Mráz, M. Plátek and J. Vogel, Restarting automata, in *FCT'95, Proc.*, edited by H. Reichel. *Lecture Notes in Computer Science* 965. Springer, Berlin (1995) 283–292.

[7] P. Jančar, F. Mráz, M. Plátek and J. Vogel, On monotonic automata with a restart operation. *J. Autom. Lang. Combin.* **4** (1999) 287–311.

[8] T. Jurdziński, K. Loryś, G. Niemann and F. Otto, Some results on RWW- and RRWW-automata and their relation to the class of growing context-sensitive languages. *J. Autom. Lang. Combin.* **9** (2004) 407–437.

[9] T. Jurdziński, F. Mráz, F. Otto and M. Plátek, Degrees of non-monotonicity for restarting automata. *Theor. Computer Sci.* **369** (2006) 1–34.

[10] T. Jurdziński and F. Otto, Shrinking restarting automata. *Int. J. Found. Comput. Sci.* **18** (2007) 361–385.

[11] T. Jurdziński, F. Otto, F. Mráz and M. Plátek, On left-monotone deterministic restarting automata, in *DLT 2004, Proc.*, edited by C.S. Calude, E. Calude and M.J. Dinneen. *Lecture Notes in Computer Science* 3340. Springer, Berlin (2004) 249–260.

[12] M. Kutrib and F. Otto, On the descriptional complexity of the window size for deleting restarting automata. *Int. J. Found. Comput. Sci.* **24** (2013) 831–846.

[13] M. Kutrib and J. Reimann, Succinct description of regular languages by weak restarting automata. *Inf. Comput.* **206** (2008) 1152–1160.

[14] K. Kwee and F. Otto, On ordered RRWW-automata, in *DLT 2016, Proc.*, edited by S. Brlek and C. Reutenauer. *Lecture Notes in Computer Science* 9840. Springer, Heidelberg (2016) 268–279.

[15] K. Kwee and F. Otto, On the effects of nondeterminism on ordered restarting automata, in *SOFSEM 2016, Proc.*, edited by R.M. Freivalds, G. Engels and B. Catania. *Lecture Notes in Computer Science* 9587. Springer, Heidelberg (2016) 369–380.

[16] K. Kwee and F. Otto, Nondeterministic ordered restarting automata. *Int. J. Found. Comput. Sci.* **29** (2018) 663–685.

[17] R. McNaughton, P. Narendran and F. Otto, Church-Rosser Thue systems and formal languages. *J. Assoc. Comput. Mach.* **35** (1988) 324–344.

[18] F. Mráz, Lookahead hierarchies of restarting automata. *J. Autom. Lang. Combin.* **6** (2001) 493–506.

[19] F. Mráz and F. Otto, Ordered restarting automata for picture languages, in *SOFSEM 2014, Proc.*, edited by V. Geffert, B. Preneel, B. Rovan, J. Štuller and A. Min Tjoa. *Lecture Notes in Computer Science* 8327. Springer, Heidelberg (2014) 431–442.

[20] F. Mráz and F. Otto. Window size two suffices for deterministic monotone RWW-automata, in *Eleventh Workshop on Non-Classical Models of Automata and Applications (NCMA 2019), Proc.*, edited by R. Freund, M. Holzer, and J.M. Sempere. volume 336 of *books@ocg.at.* Österreichische Computer Gesellschaft, Wien, (2019) 139–154.

[21] F. Mráz and F. Otto. On shrinking restarting automata of window size one and two, in *DLT 2019, Proc.*, edited by P. Hofman and M. Skrzypczak. *Lecture Notes in Computer Science* 11647. Springer, Heidelberg (2019) 140–153.

[22] B. Nagy, On a hierarchy of $5' \to 3'$ sensing Watson-Crick finite automata languages. *J. Logic Comput.* **23** (2013) 855–872.

[23] G. Niemann and F. Otto, Restarting automata, Church-Rosser languages, and confluent internal contextual languages. Mathematische Schriften Kassel 4/99, Universität Kassel (1999).

[24] G. Niemann and F. Otto, Restarting automata, Church-Rosser languages, and representations of r.e. languages, in *Developments in Language Theory – Foundations, Applications, and Perspectives, DLT 1999, Proc.*, edited by G. Rozenberg and W. Thomas. World Scientific, Singapore (2000) 103–114.

[25] G. Niemann and F. Otto, The Church-Rosser languages are the deterministic variants of the growing context-sensitive languages. *Inf. Comput.* **197** (2005) 1–21.

[26] F. Otto and T. Jurdziński, On left-monotone restarting automata. Mathematische Schriften Kassel 17/03, Universität Kassel (2003).

[27] N. Schluter, On lookahead hierarchies for monotone and deterministic restarting automata with auxiliary symbols (Extended abstract), in *DLT 2010, Proc.*, edited by Y. Gao, H. Lu and S. Seki. *Lecture Notes in Computer Science* 6224. Springer, Berlin (2010) 440–441.

[28] N. Schluter, Restarting automata with auxiliary symbols and small lookahead, in *LATA 2011, Proc.*, edited by A.H. Dediu, S. Inenaga and C. Martín-Vide. *Lecture Notes in Computer Science* 6638. Springer, Berlin (2011) 499–510.

[29] N. Schluter, Restarting automata with auxiliary symbols restricted by lookahead size. *Int. J. Computer Math.* **92** (2015) 908–938.

[30] B. v. Braunmühl and R. Verbeek, Finite-change automata, in *4th GI Conference, Proc.*, edited by K. Weihrauch. *Lecture Notes in Computer Science* 67. Springer, Berlin (1979) 91–100.